目　　录

第 1 章　概　　论

1.1　MATLAB 的发展历程和影响

MATLAB 名字由 MATrix 和 LABoratory 两词的前三个字母组合而成。那是 20 世纪 70 年代后期的事：时任美国新墨西哥大学计算机科学系主任的 Cleve Moler 教授出于减轻学生编程负担的动机，为学生设计了一组调用 LINPACK 和 EISPACK 库程序的"通俗易用"的接口，此即用 FORTRAN 语言编写的萌芽状态的 MATLAB 。

经几年的校际流传，在 Little 的推动下，由 Little、Moler、Steve Bangert 合作，于 1984 年成立了 MathWorks 公司，并把 MATLAB 正式推向市场。从这时起，MATLAB 的内核采用 C 语言编写，而且除原有的数值计算能力外，还新增了数据视图功能。

MATLAB 以商品形式出现后，仅短短几年，就以其良好的开放性和运行的可靠性，使原先控制领域里的封闭式软件包(如英国的 UMIST，瑞典的 LUND 和 SIMNON，德国的 KEDDC)纷纷淘汰，而改以 MATLAB 为平台加以重建。在时间进入 20 世纪 90 年代的时候，MATLAB 已经成为国际控制界公认的标准计算软件。

到 90 年代初期，在国际上三十几个数学类科技应用软件中，MATLAB 在数值计算方面独占鳌头，而 Mathematica 和 Maple 则分居符号计算软件的前两名。Mathcad 因其提供计算、图形、文字处理的统一环境而深受中学生的欢迎。

MathWorks 公司于 1993 年推出 MATLAB 4.0 版本，从此告别 DOS 版。4.x 版在继承和发展其原有的数值计算和图形可视能力的同时，出现了以下几个重要变化：(1)推出了 Simulink。这是一个交互式操作的动态系统建模、仿真、分析集成环境。它的出现使人们有可能考虑许多以前不得不做简化假设的非线性因素、随机因素，从而大大提高了人们对非线性、随机动态系统的认知能力。(2)开发了与外部进行直接数据交换的组件，打通了 MATLAB 进行实时数据分析、处理和硬件开发的道路。(3)推出了符号计算工具包。1993 年 MathWorks 公司从加拿大滑铁卢大学购得 Maple 的使用权，以 Maple 为"引擎"开发了 Symbolic Math Toolbox 1.0。MathWorks 公司此举加快结束了国际上数值计算、符号计算孰优孰劣的长期争论，促成了两种计算的互补发展新时代。(4)构作了 Notebook 。MathWorks公司瞄准应用范围最广的 Word，运用 DDE 和 OLE，实现了 MATLAB 与 Word 的无缝连接，从而为专业科技工作者创造了融科学计算、图形可视、文字处理于一体的高水准环境。

1997 年仲春，MATLAB 5.0 问世，紧接着陆续推出 5.1、5.2 和 1999 年春的 5.3 版。与 4.x 相比，现今的 MATLAB 拥有更丰富的数据类型和结构、更友善的面向对象、更加快速精良的图形可视、更广博的数学和数据分析资源、更多的应用开发工具(关于 MATLAB 5.x 的特点下节将作更详细的介绍)。

诚然,到1999年底,Mathematica也已经升到4.0版,它特别加强了以前欠缺的大规模数据处理能力。Mathcad也赶在2000年到来之前推出了Mathcad 2000,它购买了Maple内核和库的部分使用权,打通了与MATLAB的接口,从而把其数学计算能力提高到专业层次。但是,就影响而言,至今仍然没有一个别的计算软件可与MATLAB匹敌。

在欧美大学里,诸如应用代数、数理统计、自动控制、数字信号处理、模拟与数字通信、时间序列分析、动态系统仿真等课程的教科书都把MATLAB作为内容。这几乎成了20世纪90年代教科书与旧版书籍的标志性区别。在那里,MATLAB是攻读学位的大学生、硕士生、博士生必须掌握的基本工具。

在国际学术界,MATLAB已经被确认为准确、可靠的科学计算标准软件。在许多国际一流的学术刊物(尤其是信息科学刊物)上,都可以看到MATLAB的应用。

在设计研究单位和工业部门,MATLAB被认作进行高效研究、开发的首选软件工具。如美国National Instruments公司信号测量、分析软件LabVIEW,Cadence公司信号和通信分析设计软件SPW等,或者直接建筑在MATLAB之上,或者以MATLAB为主要支撑。又如HP公司的VXI硬件,TM公司的DSP,Gage公司的各种硬卡、仪器等都接受MATLAB的支持。

1.2 MATLAB 6.0的基本组成和特点

经过近20年实践,人们已经意识到:MATLAB作为计算工具和科技资源,可以扩大科学研究的范围、提高工程生产的效率、缩短开发周期、加快探索步伐、激发创造活力。那么,作为当前最新版本的MATLAB 6.0究竟包括哪些内容?有哪些特点呢?

1.2.1 MATLAB的语言部分

MATLAB 5.0以前版本的MATLAB语言比较简单。它只有双精度数值和简单字符串两种数据类型,只能处理1维、2维数组。它的控制流和函数形式也都比较简单。这一方面与当时软件的整体水平有关,另一方面与MATLAB仅限于数值计算和图形可视应用的设计目标有关。

从MATLAB 5.0起,MATLAB对其语言进行了根本性的变革,使之成为一种高级的“阵列”式语言。

1. MATLAB语言的传统优点

MATLAB自问世起,就以数值计算称雄。MATLAB进行数值计算的基本处理单位是复数数组(或称阵列),并且数组维数是自动按照规则确定的。这一方面使MATLAB程序可以被高度“向量化”,另一方面使用户易写易读。

比如已知t的采样数据是$(n \times m)$维数组,要计算$y = e^{-2t}\sin(5t)$。对一般的计算语言来说,必须采用两层循环才能得到结果。这不但程序复杂,而且那讨厌的循环十分费时。MATLAB处理这类问题则简洁快捷得多,它只需直截了当的一条指令y = exp(- 2 * t). * sin(5 * t),就可获得同样是$(n \times m)$维的y数组。这就是所谓的“数组运算”。这种运算在信号处理和图形可视中,将被频繁使用。

又如对于求解 $Ax=b$ 代数方程问题。教科书的基本叙述：当 A 是标量时，$x=\frac{b}{A}$；当 A 是非奇异矩阵时，$x=A^{-1}b$；当 A 是行数大于列数的满秩阵时，$x=(A^{T}A)^{-1}A^{T}b$；当 A 的列数大于行数时，x 有无数解。一般程序就必须按以上不同情况进行编程。然而对 MATLAB 来说，它只需一条指令：x = A \ b 。指令是简单的，但其内涵却远远超出了普通教科书的范围，其计算的快速性、准确性和稳定性都是普通程序所远不及的。

2. MATLAB 6.0 语言新特点

(1)数据类型和面向对象编程技术

MATLAB 6.0 与旧版最显著的不同在于数据类型的变化。MATLAB 6.0 现有六种基本数据类型：双精度数组、字符串数组、元胞数组、构架数组、稀疏矩阵和 unit8 数据。

数据类型的变革，面向对象编程技术的采用，所产生的影响之一是广泛而深层的。这种影响首先表现在 MATLAB 的自身。从 5.0 版起，MATLAB 就用新数据类型逐步地对其自身的函数指令加以改造。这个过程一直延续到 5.3 版才基本完成。比如 5.3 版就推出了一组名称全新(求取极小值等)的泛函指令，它们优化参数的设置是采用构架数组进行的。再如 5.x 版提供的常微分方程解算指令 ODE Solver 的参数设置也全是靠新数据类型进行的。

新数据类型和面向对象技术的影响之二：若干通用工具包的相应升级。以符号计算为例，在 MATLAB 4.2c 中，Symbolic Math Toolbox 1.0 处理符号计算的指令形式与数值计算指令形式很不协调，显得十分生硬。比如，符号矩阵的"四则"运算的旧版指令分别是 symadd, symsub, symmul 和 inverse 。但在 5.x 版中，符号工具包已升级为 2.0 版，新的"四则"符号运算指令形式上与数值计算完全相同，它们分别是 +，-，*，/。新的符号计算形式已被改造得与"MATLAB 风格数值计算形式"浑然统一。

新数据类型和面向对象技术影响之三：一系列的应用工具包相继升级。这不仅使应用工具包表现更为友善，而且功能大大加强。以控制工具包为例，新版利用构架数组和重载技术，把线性时不变系统(Linear Time - Invariant system)设计为"LTI 对象"。这样处理后，不管 LTI 是由传递函数产生、由零极点增益方式产生，还是由状态方程形式创建，只要是 LTI 对象，它们之间就可方便地进行各种数学运算。比如在控制教科书上，由前向控制环节 G_1、对象 G_0、反馈环节 G_2 组成的负反馈闭环系统传递函数 $G=\frac{G_0G_1}{1+G_0G_1G_2}$。假如 G_0、G_1、G_2 都是用传递函数，那么读者"手算"时，必须进行多项式展开、乘法、合并、简化等运算。如果利用"旧版 MATLAB "求 G，那末必须先使用 series 指令把 G_0、G_1 串接成"中间"函数 G_{01}，然后再用 feedback 指令由 G_{01}、G_2 算得 G。然而"新版 MATLAB "只需直截了当地用一条指令 G = G0 * G1/(1 + G0 * G1 * G2)，并且式中的 G0, G1, G2 的表达形式可以各不相同，任取传递函数、零极点增益、状态方程等形式。当然 LTI 的优点远不止于此，比如它还可直接对多输入多输出系统进行统一运算，而无须分解成若干个子系统进行。

(2)控制流和函数类型

新版 MATLAB 的控制流新增了多分支结构 switch - case、try - catch 结构和警告提示指令 error、warning，进一步提高了程序的可读性和运行可靠性。

新版的函数类型很丰富,适应编制和管理复杂程度不同的程序。例如内联函数比较简练,适用于各类比较简单数学模型。而子函数、私用函数的增添,使得复杂函数比较容易组织,既提高了软件的“重用度”,又避免了众多内存变量名的冲突和庞大工具库的函数名冲突。

为函数设计了新的变长度输入输出宗量 varagin、varagout。采用了这种变长度宗量,MATLAB自身的新版指令被进一步“柔性化”。一个指令可以接受任意多个输入宗量,可以产生任意多个输出宗量,以适应不同场合的需要。所有这些措施使得MATLAB能更加便捷地编制复杂的大型程序。当然,用户也可以借助这种变长度宗量来编制灵活多变的应用程序。

1.2.2 MATLAB 的工作环境

所谓工作环境是指:帮助系统、工作内存管理、指令和函数管理、搜索路径管理、操作系统、程序调试和性能剖析工具等。

1. 传统工作环境

与同时期其他数学类软件相比,旧版MATLAB的工作环境虽属比较友善之列,但其工作环境确实比较“单调”。它的帮助系统是“纯文本”形式的;内存管理、路径管理、调试工具是单纯指令操纵形式的;文件类型也形式单一,仅有M文件和MAT文件。4.2c版情况开始变化,但那只是过渡形式。

2. MATLAB 6.0 工作环境新特点

(1)大量引入图形用户界面

MATLAB 6.0改变了过去单调依靠“在指令窗通过纯文本形指令进行各种操作”面貌,引入了许多让使用者一目了然的图形界面,如在线帮助的交互型界面helpwin,管理工作内存的workspace,交互式的路径管理界面pathtool,指令窗显示风格设置界面等。它们的开启方式有:工具条图标开启、选择菜单项开启,直接“文本式”指令开启。

MATLAB 6.0更进一步把图形显示窗改造成了交互操作的可编辑图形界面。

(2)引入了全方位帮助系统

“临场”在线帮助:这些帮助内容,大多嵌附在M文件中,即时性强,反应速度快。它对求助内容的回答最及时准确。MATLAB旧版就一直采用这种帮助系统,并深受用户欢迎。新版保留原功能的同时,还新增一个内容与之完全对应的图形界面helpwin,加强了对用户的引导。

综合型在线帮助文库helpdesk:该文库以HTML超文本形式独立存在。整个文库按MATLAB的功能和核心内容编排,系统性强,且可以借助“超链接”方便地进行交叉查阅。但是,这部分内容偶而会发甥与真实M文件脱节的现象。

完整易读的PDF文档:这部分内容与HTML帮助文库完全对应。PDF文档不能直接从指令窗中开启,而必须借助Adobe Acrobat Reader软件阅读。这种文件的版面清楚、规范,适宜有选择地系统阅读,也适宜于制作硬拷贝。

演示软件demo:这是一个内容广泛的演示程序。MATLAB一向重视演示软件的设计,因此无论MATLAB旧版还是新版,都随带各自的演示程序。但是,新版内容更丰富了。

(3)M 文件编辑、调试的集成环境

新的编辑器有十分良好的文字编辑功能,它可采用色彩和制表位醒目地区分标识程序中不同功能的文字,如运算指令、控制流指令、注释等。通过编辑器的菜单选项可以对编辑器的文字、段落等风格进行类似 Word 那样的设置。

从 5.2 版起,还新增了"变量现场显示"功能,只要把鼠标放在变量名上(Mouse over),就能在现场显示该变量的内容。

在 MATLAB 6.0 中,调试器已经被图形化,它与编辑器集成为一体,只需点动交互窗上的调试图标就可完成对程序的调试。

(4)M 文件的性能剖析

调试器只负责 M 文件中语法错误和运行错误的定位,而性能剖析指令 profile 将给出程序各环节的耗时分析报告。MATLAB 6.0 剖析指令的分析报告特别详细,它将帮助用户寻找影响程序运行速度的"瓶颈"所在,以便改进。

(5)Notebook 新的安装方式

从 4.2c 版引入 Notebook 以来,这种集文字、计算、图形于一体的"活"环境就深受用户赞赏。但直到 5.2 版,Notebook 的安装都是与 MATLAB 的安装同步进行的。这种安装方式的不便之处是:一旦 Word 发生变动,就必须把 MATLAB 全盘重装。MATLAB 6.0 改变了这种局面,它可以在 MATLAB 指令窗中"随时"进行 Notebook 安装,省时灵活。

(6)MATLAB 环境可运行文件的多样化

旧版中,用户可编制和运行的程序文件只有 M 脚本文件和 M 函数文件。5.x 版新增了产生伪代码 P 文件的 pcode 指令和产生二进制 MEX 文件的 mex 指令。较之 M 文件,这两种文件的运行速度要快得多,保密性也好。

1.2.3 MATLAB 的视图系统

1. 传统的图形表现力

MATLAB 的图形可视能力在所有数学软件中是首屈一指的。MATLAB 的图形系统有高层和低层两个部分组成。高层指令友善、简便;低层指令细腻、丰富、灵活。

一般说来,不管二元函数多么复杂,它的三维图形,仅需 10 条左右指令,就能得到富于感染力的表现。数据和函数的图形可视手段包括:线的勾画、色图使用、浓淡处理、视角选择、透视和裁剪。MATLAB 有比较完备的图形标识指令,它们可标注:图名、轴名、解释文字和绘画图例。

2. MATLAB 6.0 的视图新特点

(1)MATLAB 6.0 的可编辑图形窗

对一般用户来说,在使用 MATLAB 6.0 图形功能时,感受最强烈的变化是图形窗。此前的图形窗只具单纯的显示功能,而 MATLAB 6.0 则不同,它是可编辑的图形显示窗。在 5.3 版的图形窗里,只需点动工具图标或菜单选项,就可直接对显示图形的各种"对象属性"进行随心所欲的设置,可交互式地改变线条型式、粗细、颜色,可动态地变换观察视角,可在图形窗随意位置标识文字或子图。MATLAB 6.0 的图形窗是十分成功的"图柄"操作的图形用户界面。

(2)MATLAB 6.0 的 Tex 特殊字符集

图形功能的另一个较大变化是标识能力的大大增强。具体表现:一,引进 Tex 特殊字符子集,可标注如 α,β,$\sum$等数学字符;二,可书写上下标;三,可对英文、中文进行字体形式和大小的设置;四,可采取多种方式进行多行文字注释。

(3)MATLAB 6.0 的简捷绘图指令

这组指令的特点是:"指令的前两个字母是 ez ",英文含义是"Easy to "。这组指令有两个功能:一,直接表现用字符串描写的函数图形;二,与符号计算配套使用,作为符号计算结果的图形可视工具。

这种指令的使用方法极其简单。例如使用一条指令 ezsurf('y/(1 + x^2 + y^2)') 就可以绘制二元函数 $z=\frac{y}{1+x^2+y^2}$的曲面。

这组指令与普通"数值型"绘图指令起着互为补充的作用。假若就方便易用排序,简捷指令最方便,普通"数值型"绘图指令次之,低层指令最繁;假若就绘图的细致和个性化能力排序,那末低层指令最强,简捷指令最弱。

(4)MATLAB 6.0 增强了高层绘图指令的排版能力

新版在同一图形窗口中可设置大小不同、非等距排列的任意个子图,而旧版只能开设面积等分的子图。新版可以在同一图形中使用两套不同的坐标系统,而旧版则不能。

(5)MATLAB 6.0 新增的其他高层绘图指令

增添了主要用于表现统计数据的面域图 area, 水平直方图 barh, 三维直方图 bar3, bar3h, 二维、三维饼图 pie, pie3, 三维杆图 stem3 等;新增了四维数值表现力更强的切片等位线图 slicecontour ;改造了切片图 slice,允许任意设置切面;新增了表现不规则数据点的三维网线和曲面图 trimesh, trisurf ;新增了若干色图函数,如 spring, summer, autumn, winter 等;增加了表现数据点的 8 种新"点型"。

(6)MATLAB 6.0 读写图像文件能力的加强

旧版 MATLAB 能读写的图像文件类型比较狭窄。新版能够读写的文件格式有: bmp, hdf, jpg, jpeg, pex, tif, tiff, xwd 等。这无疑为进一步开拓图像处理方面的应用程序提供了更好的条件。

(7)MATLAB 6.0 低层指令结构的改变和能力的加强

"轴"对象上新增的"子对象"light 。该对象的增设,再配合增强了的光照模式 lighting 和控制光线反射的材质指令 material,使得图形表现具真实感。

"轴"新增的照相机属性和投影属性,能更好地满足人们的视觉要求。

"面"对象的面色属性可以采用纹理影射技术,从而可以在各种形状曲面上彩绘各种图像,或表现表面的凹凸不平、材料纹路。

(8) MATLAB 6.0 的图形用户界面 GUI 制作工具

从 4.2c 版起,MATLAB 就开始向用户提供制作 GUI 的指令,但十分稚嫩。随 5.0, 5.1, 5.2 版的升级,GUI 制作工具不断改进。现在 MATLAB 6.0 中:不仅可以制作位置固定的用户菜单 uimenu,而且可以制作位置不固定的"现场"菜单(Context menu);用户控件 uicontrol 已增加到 10 种;不管是菜单,还是控件,都可以进行"使能"和"可见性"控制。

MATLAB 向用户提供两种制作 GUI 的途径:依靠指令制作 GUI;借助交互式工具

guide 制作 GUI 。这两种方法各有优缺点:前者灵活、细致;后者直观、全局观念强。用户交替运用这两种制作手段,可高效地制作 GUI,开发出各种生动活泼的应用程序。

1.2.4 MATLAB 的数学函数库

1. 世界一流水平的数值计算函数库

MATLAB 自问世起,就抱定一个宗旨:其所有数值计算算法都必须是国际公认的、最先进的、可靠算法;其程序由世界一流专家编制,并经高度优化;而执行算法的指令形式则必须简单、易读易用。MATLAB 正是仰赖这些高质量的数值计算函数赢得了声誉。

MATLAB 数值计算函数库的另一个特点是其内容的基础性和通用性。它正由于这一特点,而适应了诸如自动控制、信号处理、动力工程、电力系统等应用学科的需要,并进而开发出一系列应用工具包。

在整个 MATLAB 的发展过程中,这数值计算函数库,从内容到形式,变化最小。

2. MATLAB 6.0 函数库的变化

(1)MATLAB 6.0 新增的常微分方程解算程序

MATLAB 6.0 MATLAB 数值计算方面的最大变化是增添了一组常微分方程数值解算程序 ODE Solver 。这组解算程序无论是算法还是软件结构都十分精良,它包含 ode23, ode45, ode113, ode23t, ode15s, ode23s, ode23tb 等不同解算指令,用以解算包括 Stiff 方程在内的各种微分方程。

MATLAB 6.0 为解 ODE 问题所设计的文件十分严整,包括解算指令 Solver、被 Solver 调用的微分方程描述文件、进行积分算法参数设置的 odeset 和 odeget 、解算输出指令 odeplot, odephas2 等。

(2)MATLAB 6.0 新增的其他数值计算指令

MATLAB 6.0 版新增的许多计算指令与数值计算方面的最新成就直接相关。举例来说,新版及时地增添了广义奇异值计算指令 gsvd ;高维快速傅里叶变换和反变换 fftn, ifftn ;高维插值指令 interpn 等。

3. MATLAB 6.0 的符号计算工具包

关于符号计算和数值计算优劣的争论曾经历过一段时间,但那是 20 世纪 90 年代以前的事。MathWorks 公司打破门户之见,把 Maple 的内核和数学函数库引入了 MATLAB,从而使 MATLAB 具有了数值和符号双重计算能力。用户可以视具体问题而进行适当的选择。比如,对于比较复杂的"初值类"非线性微分方程,有时符号计算或无法解、或求解时间太长,而数值算法却比较有效;反之,对于边值类微分方程,数值算法的实现可能比较繁琐,而符号计算有时倒比较简便。

Maple 的函数库十分庞大,包含 2000 多个函数。它几乎囊括了一般用户所需的所有函数。与 5.x 版 MATLAB 配用的 Symbolic Math Toolbox 2.0 允许用户在三个不同层次上做符号计算。第一层次是,在进行符号对象定义后,直接利用 MATLAB 格式进行矩阵分解、微分、积分、积分变换、代数方程求解、微分方程求解等运算。第二层次是,借助 Maple 指令,把单个 Maple 格式的指令送往 Maple 引擎计算。第三层次是,借助 procread 把整段 Maple 程序送往 Maple 计算。

1.2.5 MATLAB 与外部程序的交互

MATLAB 的卓越性能引发了用户的新需求：希望把在 MATLAB 环境中编制的程序开发成能脱离 MATLAB 独立运行的程序；希望能在外部程序中调用 MATLAB 作为计算引擎。需求决定商品，市场也真出现了诸如 Mediva 等商品软件，能把 MATLAB 的 M 文件转变为独立于平台的 EXE 可执行文件；出现许多专用软件把 MATLAB 直接当计算引擎使用。鉴于此，MATLAB 在 4.x 版时代的后期，就尝试性地推出了自己的编译器。

1. 与 MATLAB 6.0 配用的编译器 Compiler 3.0

伴随 MATLAB 5.0 向 MATLAB 6.0 升级的过程中，变化较大，更新较快的是 MATLAB 编译器。与 5.2 版配用的是 Compiler 1.2 版，而与 MATLAB 6.0 配用的则是 Compiler3.0 版、2.0 版。

无论是 1.2 版编译器，还是 2.0 版编译器，它们都不但可以把全 M 函数文件编译成独立应用程序，而且也可以把 C 或 FORTRAN 程序与 M 文件混编成独立应用程序。这种程序的优点是：一，可以脱离 MATLAB 环境独立运行；二，运行速度快。

MATLAB 编译器无疑给用户开发计算类 EXE 可执行程序提供了快捷、高效的工具。举例来说，假如用户想编制一个求解各类线性代数方程的可独立执行的程序，用户只要先在 MATLAB 环境中编写由 30 来条指令组成的 M 函数文件，然后借助编译器把它变成独立执行的 EXE 程序。这 EXE 文件不但可以计算"恰定"方程，而且可以解算"超定"、"欠定"方程。值得指出：原 M 文件是不过 2K 字节的小小程序，而编译生成的 EXE 文件却超过 200K 字节，这可是一个不算小的程序。

与 1.2 版相比，3.0 版编译器具有处理高维数组、元胞数组、构架数组的能力，支持变长度输入输出宗量，支持多分支等控制流。

2. MATLAB 6.0 的 API 应用程序接口

与 MATLAB 编译器相比，MATLAB 的 API 应用程序接口问世得更晚，也更不成熟。MATLAB API 由一系列接口指令组成。借助这些接口指令，用户就可在 C 或 FORTRAN 中，或直接读写 MATLAB 的 MAT 数据文件，或把 MATLAB 当作计算引擎使用。

1.3 与 MATLAB 6.0 配用的 Simulink 3.0

Simulink 是 MathWorks 公司开发的又一个产生重大影响的软件产品。它的前身 SIMULIB 问世于 20 世纪 90 年代初，以工具库的形式挂接在 MATLAB 3.5 上。以 Simulink 名称广为人知，是在 MATLAB 4.2x 时期。Simulink 不能独立运行，而只能在 MATLAB 环境中运行。现在较为流行的有：与 MATLAB 5.2 配用的 Simulink 2.2；与 MATLAB 5.3 配用的 Simulink 3.0。

1.3.1 Simulink 的传统优点

不管是什么版本，Simulink 总由模块库、模型构造及分析指令、演示程序等三部分组成。在 Simulink 环境中，对于由微分方程或差分方程描写的动态系统，用户无须编写文本形式的程序，而只要通过一些简单的鼠标操作就可形象地建立起被研究系统的数学模

型,并进行仿真和分析研究。

举例来说,面对一个由微分方程描写的动态系统,用户有如下三个研究途径:一,直接利用 ODE Solver 数值解算指令编写表示那系统的 M 文件;二,利用符号计算指令编写相应的程序;三,在 Simulink 环境中建立那系统的方块图模型。三者比较而言,Simulink 是最合适、最方便、最直观的研究环境。在 Simulink 中,那些以往不得不忽略的非线性、随机干扰等因素的影响也十分容易研究。

1.3.2 Simulink 3.0 的特点

(1)Simulink 系列软件产品

经过几年的努力,MathWorks 公司已经把 Simulink 发展成一个系列产品。例如,它与 Stateflow 状态流配合,可以建立更清晰的离散事件系统的概念化模型;与 Real - Time Workshop 配合可产生进行实时仿真和运行于各种硬件的 C 码;与 DSP Blockset 配用可以进行 DSP 装置和系统的快速设计和仿真。Simulink 在 Communication Toolbox、Nonlinear Control Design Blockset、Power System Blockset 等专业工具包的配合下,就可对通信系统、非线性控制系统、电力系统进行深入的建模、仿真和分析研究。

(2)Simulink 3.0 的模型库

与以前版本相比,Simulink 3.0 本体模型库的结构已彻底改变。原先是子库分层的块图结构,现在是树状的文件夹形式,并且旧版中只有部分子库与新版文件夹对应。

与 Simulink 3.0 配用的工具包显著增加。新版除把原 2.2 版 12 个应用子库改造为 12 个文件夹之外,又增添了 2 个新文件夹。

(3)模块的“使能”和“触发”功能

从 2.x 版起,Simulink 增设了“使能”和“触发”功能。这就为 Simulink 进行离散事件系统的仿真打下了基础。借助“使能”和“触发”功能,用户可以建立各种根据状态组合和迁移改变模型结构的复杂系统。

第2章　M文件和面向对象编程

使用 MATLAB 函数时,例如 inv, abs, angle 和 sqrt,MATLAB 获取传递给它的变量,利用所给的输入,计算所要求的结果。然后,把这些结果返回。由函数执行的命令,以及由这些命令所创建的中间变量,都是隐含的。所有可见的东西是输入和输出,也就是说函数是一个黑箱。

这些属性使得函数成为强有力的工具,用以计算命令。这些命令包括在求解一些大的问题时,经常出现的有用的数学函数或命令序列。由于这个强大的功能,MATLAB 提供了一个创建用户函数的结构,并以 M 文件的文本形式存储在计算机上。MATLAB 函数 fliplr 是一个 M 文件函数良好的例子。

```
function y = fliplr(x)
%   FLIPLR   Flip matrix in the left/right direction.
%   FLIPLR(X) returns X with row preserved and columns flipped
%   in the left/right direction.
%   X = 1  2  3  becomes  3  2  1
%   4  5  6               6  5  4
%   See also FLIPUD, ROT90.

%   Copyright (c) 1984 - 94 by The MathWorks, Inc.

[m, n] = size(x);
y = x(: , n : -1 : 1);
```

一个函数 M 文件与脚本文件类似之处在于它们都是一个有 .m 扩展名的文本文件。如同脚本 M 文件一样,函数 M 文件不进入命令窗口,而是由文本编辑器所创建的外部文本文件。一个函数的 M 文件与脚本文件在通信方面是不同的。函数与 MATLAB 工作空间之间的通信,只通过传递给它的变量和通过它所创建的输出变量。在函数内中间变量不出现在 MATLAB 工作空间,或与 MATLAB 工作空间不交互。正如上面的例子所看到的,一个函数的 M 文件的第一行把 M 文件定义为一个函数,并指定它的名字。它与文件名相同,但没有 .m 扩展名。它也定义了它的输入和输出变量。接下来的注释行是所展示的文本,它与帮助命令:" help fliplr 相对应。第一行帮助行称为 H1 行,是由 lookfor 命令所搜索的行。最后,M 文件的其余部分包含了 MATLAB 创建输出变量的命令。

假如读者想灵活运用 MATLAB 去解决实际问题,想充分调动 MATLAB——科学技术资源,想理解 MATLAB 版本升级所依仗的基础,那么本章内容将十分有用。

本章将涉及比较深层的 MATLAB 内容:脚本;函数(一般函数、内联函数、子函数、私

用函数、方法函数);程序调试和剖析;数据结构(类、对象);重载和继承;面向对象编程。本章配备了许多精心设计的算例。这些算例是完整的,可直接演练的。读者通过这些算例,将真切感受到抽象概念的内涵、各指令间的协调,将从感知上领悟到面向对象编程的优越和至关要领。

2.1 入　门

2.1.1 编写和运行

先看一个例子。通过M脚本文件,画出下列分段函数所表示的曲面。

$$p(x_1,x_2)=\begin{cases}0.5457e^{-0.75x_2^2-3.75x_1^2-1.5x_1} & x_1+x_2>1\\ 0.7575e^{-x_2^2-6x_1^2} & -1<x_1+x_2\leqslant 1\\ 0.5457e^{-0.75x_2^2-3.75x_1^2+1.5x_1} & x_1+x_2\leqslant -1\end{cases}$$

(1) 编写M脚本文件的步骤

• 点击MATLAB指令窗工具条上的New File图标,就可打开如图2-1所示的MATLAB文件编辑调试器MATLAB Editor/Debugger。其窗口名为untitled,用户即可

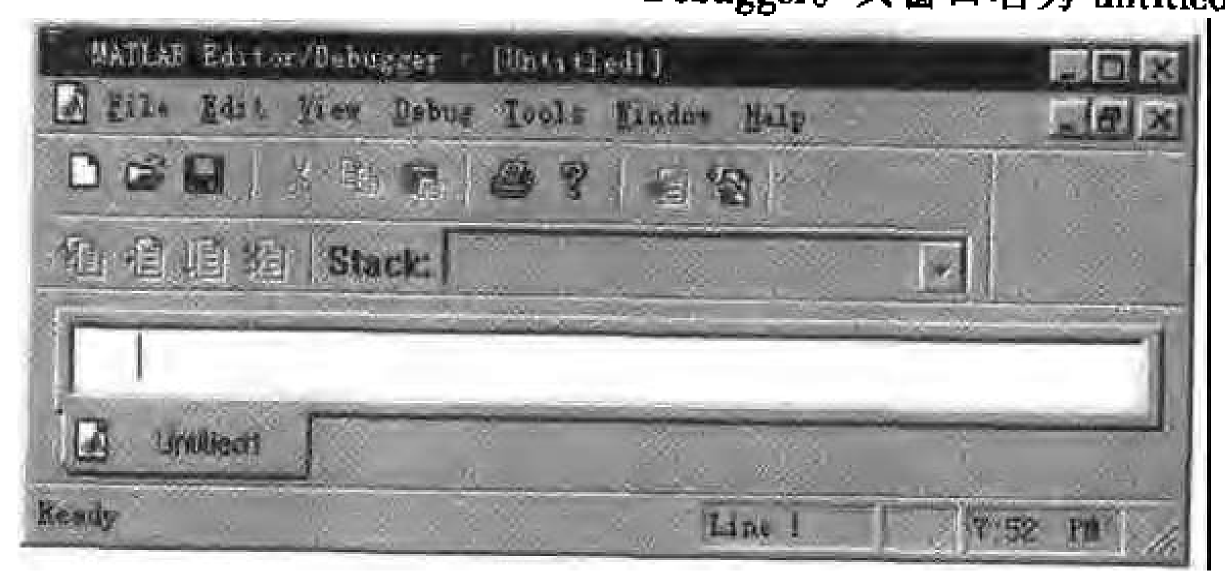


图2-1 MATLAB Editor/Debugger窗口

在空白窗口中编写程序。比如输入如下一段程序:

```
[zx81.m]
%zx81.m    This is my first example.                          <1>
a=2;b=2;                                              %       <2>
clf;
x=-a:0.2:a;y=-b:0.2:b;
for i=1:length(y)
  for j=1:length(x)
    if x(j)+y(i)>1
      z(i,j)=0.5457*exp(-0.75*y(i)^2-3.75*x(j)^2-1.5*x(j));
    elseif x(j)+y(i)<=-1
      z(i,j)=0.5457*exp(-0.75*y(i)^2-3.75*x(j)^2+1.5*x(j));
    else z(i,j)=0.7575*exp(-y(i)^2-6.*x(j)^2);
```

```
    end
  end
end
axis([-a,a,-b,b,min(min(z)),max(max(z))]);
colormap(flipud(winter));surf(x,y,z);
```

• 点击编辑调试器工具条图标▇,在弹出的 Windows 标准风格的“保存为”对话框中,选择保存文件夹,键入新编文件名(如 zx81),点动【保存】键,就完成了文件保存。

(2)运行文件

• 使 zx81.m 所在目录成为当前目录,或让该目录处在 MATLAB 的搜索路径上。

• 然后运行以下指令,便可得到如图 2-2 所示的图形。

```
zx81
```

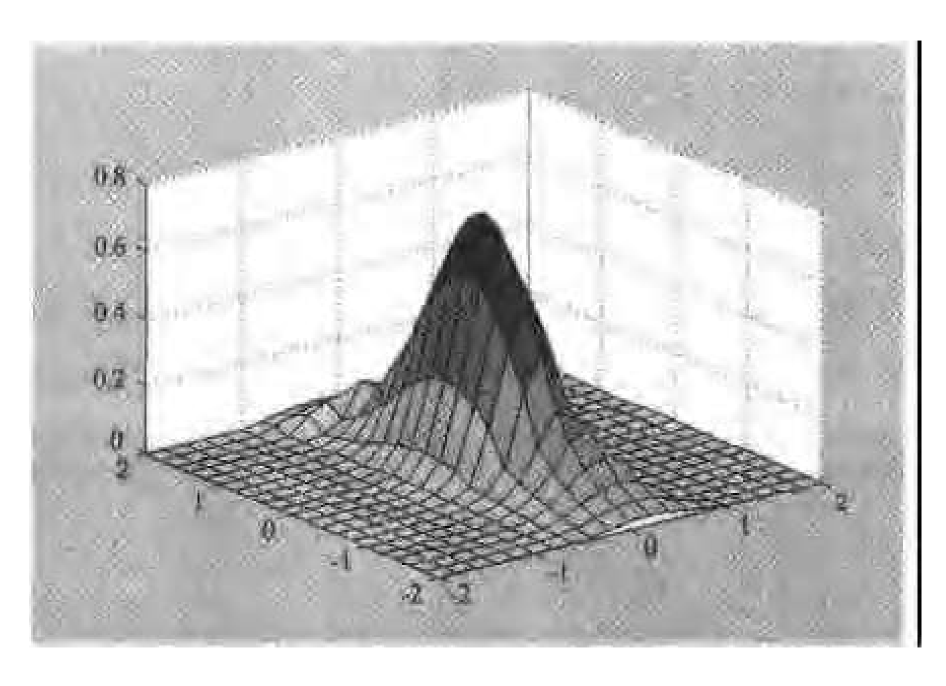


图 2-2 运行 zx81.m 得到的图形

通过 M 函数文件画出上例分段函数的曲面。

整个编程步骤和上例相同。在此演示,如何在 zx81.m 基础上产生函数文件 zx82.m。

• 在编辑调试器中,选择【File:Save As】子菜单,把 zx81.m 文件“另存为”zx82.m 。

• 用下面 4 行指令代替原文件的第 <1> <2> 条指令。

```
function  zx82(a, b)
% This is my second example.
% a   Define the limit of variable x .
% b   Define the limit of variable y .
```

• 进行上述修改后,对 zx82.m 再次实施“保存”操作。

• 在 MATLAB 指令窗中,运行以下指令,就能产生与图 2-2 完全相同的图形。

```
>>zx82(2,2)
```

2.1.2 规则和属性

M 文件函数必须遵循以下特定的规则。除此之外,它们有许多的重要属性。包括:

(1) 函数名和文件名必须相同。例如,函数 fliplr 存储在名为 fliplr.m 文件中。

(2) MATLAB 头一次执行一个 M 文件函数时,它打开相应的文本文件并将命令编

辑成存储器的内部表示，以加速执行以后所有的调用。如果函数包含了对其他 M 文件函数的引用，它们也同样被编译到存储器。普通的脚本 M 文件不被编译，即使它们是从函数 M 文件内调用；打开脚本 M 文件，调用一次就逐行进行注释。

(3) 在函数 M 文件中，到第一个非注释行为止的注释行是帮助文本。当需要帮助时，返回该文本。例如，>> help fliplr 返回上述前八行注释。

(4) 第一行帮助行，名为 H1 行，是由 lookfor 命令搜索的行。

(5) 函数可以有零个或更多个输入参量。函数可以有零个或更多个输出参量。

(6) 函数可以按少于函数 M 文件中所规定的输入和输出变量进行调用，但不能用多于函数 M 文件中所规定的输入和输出变量数目。如果输入和输出变量数目多于函数 M 文件中 function 语句一开始所规定的数目，则调用时自动返回一个错误。

(7) 当函数有一个以上输出变量时，输出变量包含在括号内。例如，[V,D] = eig(A)。不要把这个句法与等号左边的[V,D] 相混淆。左边的[V,D] 是由数组 V 和 D 所组成。

(8) 当调用一个函数时，所用的输入和输出的参量的数目，在函数内是规定好的。函数工作空间变量 *nargin* 包含输入参量个数；函数工作空间变量 nargout 包含输出参量个数。事实上，这些变量常用来设置缺省输入变量，并决定用户所希望的输出变量。例如，考虑 MATLAB 函数 linspace：

```
function y = linspace(d1, d2, n)
%   LINSPACE Linearly spaced vector.
%    LINSPACE(x1, x2) generates a row vector of 100 linearly
%    equally spaced points between x1 and x2.
%    LINSPACE(x1, x2, N) generates N points between x1 and x2.
%    See also LOGSPACE,:.


%    Copyright (c) 1984 - 94 by The MathWorks, Inc.


if nargin = = 2
  n = 100;
end
y = [d1 + (0:n - 2) * (d2 - d1)/(n - 1) d2];
```

这里，如果用户只用两个输入参量调用 linspace，例如 linspace(0,10)，linspace 产生 100 个数据点。相反，如果输入参量的个数是 3，例如，linspace(0,10,50)，第三个参量决定数据点的个数。

可用一个或两个输出参量调用的函数的一个例子是 MATLAB 函数 size。尽管这个函数不是一个 M 文件函数(它是一个内置函数)，但 size 函数的帮助文本说明了它的输出参量的选择。

```
SIZE   Matrix dimensions.
   D = SIZE(X), for M - by - N matrix X, returns the two - element
   row vector D = [M, N] containing the number of rows and columns
   in the matrix.
```

```
[M, N] = SIZE(X) returns the number of rows and columns
in separate output variables.
```

如果函数仅用一个输出参量调用,就返回一个二元素的行,它包含行数和列数。相反,如果出现两个输出参量,size 分别返回行和列。在 M 文件函数里,变量 nargout 可用来检验输出参量的个数,并按要求修正输出变量的创建。

(9) 当一个函数说明一个或多个输出变量,但没有要求输出时,就简单地不给输出变量赋任何值。MATLAB 函数 toc 阐明了这个属性。

```
function t = toc
%   TOCRead the stopwatch timer.
%   TOC, by itself, prints the elapsed time since TIC was used.
%   t = TOC; saves the elapsed time in t, instead of printing it out.
%   See also TIC, ETIME, CLOCK, CPUTIME.

%   Copyright (c) 1984 - 94 by The MathWorks, Inc.

%   TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if nargout < 1
    elapsed _ time = etime(clock, TICTOC)
else
    t = etime(clock, TICTOC);
end
```

如果用户用不以输出参量调用 toc ,例如,≫toc,就不指定输出变量 t 的值,函数在命令窗口显示函数工作空间变量 elapsed _ time ,但在 MATLAB 工作空间里不创建变量。相反,如果 toc 是以 >> out = toc 调用,则按变量 out 将消逝的时间返回到命令窗口。

(10) 函数有它们自己的专用工作空间,它与 MATLAB 的工作空间分开。函数内变量与 MATLAB 工作空间之间唯一的联系是函数的输入和输出变量。如果函数任一输入变量值发生变化,其变化仅在函数内出现,不影响 MATLAB 工作空间的变量。函数内所创建的变量只驻留在函数的工作空间,而且只在函数执行期间临时存在,以后就消失。因此,从一个调用到下一个调用,在函数工作空间变量存储信息是不可能的。如下所述,使用全局变量就提供这个特征。

(11) 如果一个预定的变量,例如,pi, 在 MATLAB 工作空间重新定义,它不会延伸到函数的工作空间。逆向有同样的属性,即函数内的重新定义变量不会延伸到 MATLAB 的工作空间中。

(12) 当调用一个函数时,输入变量不会拷贝到函数的工作空间,但使它们的值在函数内可读。然而,改变输入变量内的任何值,那么数组就拷贝到函数工作空间。进而,按缺省,如果输出变量与输入变量相同,例如,函数 x = fun(x, y, z) 中的 x ,那么就将它拷贝到函数的工作空间。因此,为了节约存储和增加速渡,最好是从大数组中抽取元素,然

后对它们作修正,而不是使整个数组拷贝到函数的工作空间。

(13) 如果变量说明是全局的,函数可以与其他函数、MATLAB 工作空间和递归调用本身共享变量。为了在函数内或 MATLAB 工作空间中访问全局变量,在每一个所希望的工作空间,变量必须说明是全局的。全局变量使用的例子可以在 MATLAB 函数 tic 和 toc 中看到,它们合在一起工作如一个跑表。

```
function tic
%   TIC Start a stopwatch timer.
%   The sequence of commands
%   TIC
%   any stuff
%   TOC
%   prints the time required for the stuff.

%   See also TOC, CLOCK, ETIME, CPUTIME.

%   Copyright (c) 1984-94 by The MathWorks, Inc.

%   TIC simply stores CLOCK in a global variable.
global TICTOC
TICTOC = clock;

function t = toc
%   TOCRead the stopwatch timer.
%   TOC, by itself, prints the elapsed time since TIC was used.
%   t = TOC; saves the elapsed time in t, instead of printing it out.
%   See also TIC, ETIME, CLOCK, CPUTIME.

%   Copyright (c) 1984-94 by The MathWorks, Inc.

%   TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if nargout < 1
  elapsed_time = etime(clock,TICTOC)
else
  t = etime(clock,TICTOC);
end
```

在函数 tic 中,变量 TICTOC 说明为全局的,因此它的值由调用函数 clock 来设定。以后在函数 toc 中,变量 TICTOC 也说明为全局的,让 toc 访问存储在 TICTOC 中的值。利用这个值,toc 计算自执行函数 tic 以来消逝的时间。值得注意的是,变量 TICTOC 存在于 tic 和 toc 的工作空间,而不在 MATLAB 工作空间。

(14) 实际编程中,无论什么时候应尽量避免使用全局变量。要是用了全局变量,建议全局变量名要长,它包含所有的大写字母,并有选择地以首次出现的 M 文件的名字开头。如果遵循建议,则在全局变量之间不必要的互作用减至最小。例如,如果另一函数或 MATLAB 工作空间说明 TICTOC 为全局的,那么它的值在该函数或 MATLAB 工作空间内可被改变,而函数 toc 会得到不同的、可能是无意义的结果。

(15) MATLAB 以搜寻脚本文件的同样方式搜寻函数 M 文件。例如,输入" cow , MATLAB 首先认为 cow 是一个变量。如果它不是,那么 MATLAB 认为它是一个内置函数。如果还不是,MATLAB 检查当前 cow.m 的目录或文件夹。如果它不存在,MATLAB 就检查 cow.m 在 MATLAB 搜寻路径上的所有目录或文件夹。如需要更多的信息,请参阅本书的 2.10 节或 MATLAB 用户指南中"MATLAB 搜寻路径"。

(16) 从函数 M 文件内可以调用脚本文件。在这种情况下,脚本文件查看函数工作空间,不查看 MATLAB 工作空间。从函数 M 文件内调用的脚本文件不必用调用函数编译到内存。函数每调用一次,它们就被打开和解释。因此,从函数 M 文件内调用脚本文件减慢了函数的执行。

(17) 函数可以递归调用。即 M 文件函数能调用它们本身。例如,考虑一个傻函数 iforgot:

```
function iforgot(n)
%   IFORGOT Recursive Function Call Example

%   Copyright (c) 1996 by Prentice - Hall, Inc

if nargin = = 0, n = 20; end
if n > 1
   disp(' I will remember to do my homework. ')
   iforgot(n - 1)
else
   disp(' Maybe NOT! ')
end
```

调用这个函数产生

```
>>iforgot(10)
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
I will remember to do my homework.
Maybe NOT!
```

递归调用函数功能在许多应用场合是有用的。在编制要递归调用的函数时,必须确保会终止,否则 MATLAB 会陷入死循环。最后,在一个递归函数内,如果变量说明是全局的,则该全局变量对以后所有函数调用是可用的。在这个意义下,全局变量变成静态的,并在函数调用之间不会消失。

(18) 当函数 M 文件到达 M 文件终点,或者碰到返回命令 return,就结束执行和返回。return 命令提供了一种结束一个函数的简单方法,而不必到达文件的终点。

(19) MATLAB 函数 error 在命令窗口显示一个字符串,放弃函数执行,把控制权返回给键盘。这个函数对提示函数使用不当很有用,如在以下文件片段中:

```
if length(val) > 1
  error('VAL must be a scalar.')
end
```

这里,如果变量 val 不是一个标量,error 显示消息字符串,把控制权返回给命令窗口和键盘。

(20) 当一个函数的输入参量的个数超出了规定的范围,MATLAB 函数 nargchk 提供了统一的响应。函数 nargchk 给定为:

```
function  msg = nargchk(low, high, number)
%  NARGCHK  Check number of input arguments.
%   Return error message if not between low and high.
%   If it is, return empty matrix.

%   Copyright (c) 1984 - 94 by The MathWorks, Inc.

msg =[  ];
if (number < low)
  msg ='Not enough input arguments.';
elseif(number  >  high)
  msg ='Too many input arguments.';
end
```

下列的文件片段表明了在一个 M 文件函数内的典型用法:

```
error(nargchk(nargin, 2, 5))
```

如上所示,如果 nargin 的值小于 2,函数 error 像前面描述的那样进行处理,nargchk 返回字符串'没有足够的输入参量'。如果 nargin 的值大于 5,函数 error 执行处理,nargchk 返回字符串'太多输入参量'。如果 nargin 是在 2 和 5 之间,函数 error 简单地将控制传递给下一个语句,nargchk 返回一个空字符串。也就是说,当它的输入参量为空,error 函数什么也不做。

(21) 当 MATLAB 运行时,它缓存了(caches)存储在 Toolbox 子目录和 Toolbox 目录内的所有子目录中所有的 M 文件的名字和位置。这使 MATLAB 很快地找到和执行函数 M 文件。也使得命令 lookfor 工作更快。被缓存的 M 文件函数当作是只读的。如果执行这些函数,以后又发生变化,MATLAB 将只执行以前编译到内存的函数,不管已改变的 M 文件。而且,在 MATLAB 执行后,如果 M 文件被加到 Toolbox 目录中,那么它们将不

出现在缓存里,因此不可利用。所以,在 M 文件函数的使用中,最好把它们存储在 Toolbox 目录外,或许最好存储在 MATLAB 目录下,直至它们被认为是完备的(complete)。当它们是完备时,就将它们移到一个只读的 Toolbox 目录或文件夹的子目录内。最后,要确保 MATLAB 搜索路径改变,以确认它们的存在。

(22) 在 Toolbox 目录外,MATLAB 跟踪 M 文件的修改日期。所以,当遇到一个以前编译到内存的 M 文件函数时,MATLAB 把已编译的 M 文件的修改日期与在磁盘上的 M 文件比较。如果日期是相同的,MATLAB 执行已编译的 M 文件。相反,如果在磁盘上的 M 文件是新的,MATLAB 清除以前已编译的 M 文件,且编译这个新的和修改过的 M 文件。

(23) M 文件的缓存过程按 MATLAB 版本而稍有不同。例如,MATLAB 4.2c 在 Macintosh 机上同样可以缓存当前的目录,因为这是第一个所搜索的磁盘位置。这个 MATLAB 版本也允许有选择地将整个 MATLAB 搜索路径缓存,并把高速缓存信息存储在一个文件中。这样,使 MATLAB 引导更快,寻找和编译所有函数 M 文件更快。退出缓存,不检测已修改的或已增加的 M 文件。当新的 M 文件加到一个缓存区时,只有当高速缓存由命令" path 刷新时,MATLAB 才能找到它们;另一方面,当修改缓存的 M 文件时,只有当以前编译过的版本由 clear 命令从内存中清除,MATLAB 才识别这个变化。例如," clear myfun,从内存中清除 M 文件函数 myfun,或" clear functions,从内存中清除所有已编译的函数。

(24) 在变量 mfilename 函数内,有要执行的 M 文件的名字。例如,正在执行 M 文件 function.m 时,函数的工作空间包含变量 mfilename,它包含函数字符串。这个变量也存在于脚本文件里,在这种情况下,它包含了要执行的脚本文件的名字。

(25) M 文件函数可像 MATLAB 命令一样工作,典型的 MATLAB 命令包括 clear, disp, echo, diary, save, hold, load, more 和 format。通常,调用一个函数把参量放在括号内。例如 size(A)。然而,如果函数有字符串参量,那么,函数可按通常函数进行调用,如,disp(' To be or not to be '),或像一个 MATLAB 命令来使用,如 clear functions。换句话说,当要求 MATLAB 解释一个表达式" command argument 时,MATLAB 认为它如同" command('argument')一样。事实上,MATLAB 命令本身能像函数那样调用!例如" format long 和" format('long')二者都把数据变成长格式。类似地," format short e 等价于" format(' short','e')。正如最后的例子所示,空格(逗号,分号)把各个命令参量分开。因此," disp How about this? 产生一个错误,因为命令 disp 只允许一个输入参量,不是三个。如果参量包含在引号里,那么 MATLAB 就忽略空格;例如," disp'How about this?'与" disp('How about this?')等价,并产生所希望的结果。

总之,函数 M 文件提供了一个简单的扩展 MATLAB 功能的方法。事实上,MATLAB 本身的许多标准函数就是 M 文件函数。

2.2 MATLAB 控制流

2.2.1 for 循环结构

一个简单的 for 循环示例。

```
for i = 1:10;                    %i 依次取 1,2,…,10。
x(i) = i;                        %对每个 i 值,重复执行由该指令构成的循环体。
end;
x                                %要求显示运行后数组 x 的值。
x =
     1     2     3     4     5     6     7     8     9    10
```

2.2.2 while 循环结构

Fibonacci 数组的元素满足 Fibonacci 规则：$a_{k+2} = a_k + a_{k+1}$,($k = 1,20\cdots$)；且 $a_1 = a_2 = 1$。现要求该数组中第一个大于 10000 的元素。

```
a(1) = 1;a(2) = 1;i = 2;
while  a(i) < = 10000
     a(i + 1) = a(i - 1) + a(i);   %当现有的元素仍小于 10000 时,求解下一个元素。
     i = i + 1;
end;
i,a(i),
i =
     21
ans =
   10946
```

2.2.3 if-else-end 分支结构

一个简单的分支结构。

```
cost = 10;number = 12;
if number > 8
   sums = number * 0.95 * cost;
end,sums
sums =
   114.0000
```

用 for 循环指令来寻求 Fibonacc 数组中第一个大于 10000 的元素。

```
n = 100;a = ones(1,n);
for i = 3:n
   a(i) = a(i - 1) + a(i - 2);
   if a(i) > = 10000
   a(i),
   break;  %跳出所在的一级循环。
    end;
end,i
ans =
```

```
10946
i =
21
```

2.2.4 switch – case 结构

学生的成绩管理,用来演示 switch 结构的应用。

```
clear;
%划分区域:满分(100),优秀(90~99),良好(80~89),及格(60~79),不及格(<60)。
for i=1:10;a{i}=89+i;b{i}=79+i;c{i}=69+i;d{i}=59+i;end;c=[d,c];
Name={' Jack','Marry','Peter',' Rose','   Tom'}; %元胞数组
Mark={72,83,56,94,100};Rank=cell(1,5);
%创建一个含 5 个元素的构架数组 S,它有三个域。
S=struct('Name',Name,'Marks',Mark,'Rank',Rank);
%根据学生的分数,求出相应的等级。
for i=1:5
  switch S(i).Marks
  case 100                          %得分为 100 时
    S(i).Rank='满分';               %列为'满分'等级
  case a                            %得分在 90 和 99 之间
    S(i).Rank='优秀';               %列为'优秀'等级
  case b                            %得分在 80 和 89 之间
    S(i).Rank='良好';               %列为'良好'等级
  case c                            %得分在 60 和 79 之间
    S(i).Rank='及格';               %列为'及格'等级
  otherwise                         %得分低于 60
    S(i).Rank='不及格';             %列为'不及格'等级
  end
end
%将学生姓名,得分,登记等信息打印出来。
disp(['学生姓名','得分','等级']);disp('   ')
for  i=1:5;
disp([S(i).Name,blanks(6),num2str(S(i).Marks),blanks(6),S(i).Rank]);end;
     学生姓名        得分        等级
     Jack            72          及格
     Marry           83          良好
     Peter           56          不及格
     Rose            94          优秀
     Tom             100         满分
```

2.2.5　try - catch 结构

try - catch 结构应用实例。

```
clear, N = 4; A = magic(3);            %设置 3 行 3 列矩阵 A
try
    A_N = A(N,:),                      %取 A 的第 N 行元素
catch
    A_end = A(end,:),                  %如果取 A(N,:)出错,则改取 A 的最后一行
end
lasterr                                %显示出错原因
A_end =
    4    9    2
ans =
Index exceeds matrix dimensions.
```

2.3　脚本文件和函数文件

2.3.1　M 文件的一般结构

M 函数文件示例。

```
[circle.m]
function sa = circle(r,s)
%CIRCLE          plot a circle of radii r in the line specified by s.
%   r            指定半径的数值
%   s            指定线色的字符串
%   sa           圆面积
%
% circle(r)       利用蓝实线画半径为 r 的圆周线
% circle(r,s)     利用串 s 指定的线色画半径为 r 的圆周线
% sa=circle(r)    计算圆面积,并画半径为 r 的蓝色圆面
% sa=circle(r,s)  计算圆面积,并画半径为 r 的 s 色圆面
% 编写于 1999 年 4 月 7 日,修改于 1999 年 8 月 27 日。
if nargin>2
   error('输入宗量太多。');
end;
if nargin==1
   s='b';
end;
clf;
```

```
t=0:pi/100:2*pi;
x=r*exp(i*t);
if nargout==0
  plot(x,s);
else
  sa=pi*r*r;
  fill(real(x),imag(x),s)
end
axis('square')
```

2.3.2 "变长度"输入输出宗量

变长度宗量使用示例。

(1)编写函数文件 ringzy.m

[ringzy.m]

```
function varargout = ringzy(r,varargin)
%RINGZY          Plot a ring and calculate the area of the ring.
%  r             基圆半径
% 调用格式
% [x1,y1,x2,y2,s1,s2]=ringzy(r,r2,'PropertyName','PropertyValue',…)
%                ①无输出时,绘圆或环。
%                ②有输出时,不绘图。
%                 (x1,y1),(x2,y2)分别是两个圆的坐标点;
%                 s1 是基圆面积;
%                 s2 为正值时,表示内环面积;为负值时,表示外环面积。
vin=length(varargin);Nin=vin+1;                               %<11>
error(nargchk(1,Nin,nargin))          %检查输入变量数目是否合适
if nargout>6                          %检查输出变量数目是否合适
   error('Too many output arguments')
end
t=0:pi/20:2*pi;x=r*exp(i*t);s=pi*r*r;
if nargout==0
   switch Nin
   case 1
plot(x,'b')
   case 2
r2=varargin{1};                                               %<22>
   x2=r2*exp(i*t);
plot(x,'b');hold on ;plot(x2,'b');hold off
   otherwise
```

```
r2 = varargin{1};                                               % <26>
   x2 = r2 * exp(i * t);
plot(x,varargin{2:end});hold on          %利用元胞数组设置对象属性    <28>
plot(x2,varargin{2:end});hold off        %利用元胞数组设置对象属性    <29>
   end;
   axis('square')
else
  varargout{1} = real(x);varargout{2} = imag(x);                % <33>
  varargout{5} = pi * r * r;varargout{6} = [   ];               % <34>
  if Nin > 1
    r2 = varargin{1};                                           % <36>
    x2 = r2 * exp(i * t);
    varargout{3} = real(x2);varargout{4} = imag(x2);            % <38>
    varargout{6} = pi * (r^2 - r2^2);                           % <39>
  end;
end
```

(2)有输出情况:请读者自己用 plot(x1,y1,x2,y2) 检验下列三个调用示例的运行结果。

```
r1 = 1;r2 = 3;
[x1,y1,x2,y2,s1,s2] = ringzy(r1);
[x1,y1,x2,y2] = ringzy(r1,r2);
[x1,y1,x2,y2,s1,s2] = ringzy(r1,r2);
```

(3)无输出情况:为节省篇幅,在此给出三个调用示例。

```
r1 = 1;r2 = 0.6;
subplot(1,3,1),ringzy(r1,r2),
subplot(1,3,2),ringzy(r1,r2,'Marker','o')
subplot(1,3,3),ringzy(r1,r2,'LineWidth',5,'Color',[1 0.4 0])
```

变长度输入宗量不同调用格式产生的图形,如图 2-3 所示。

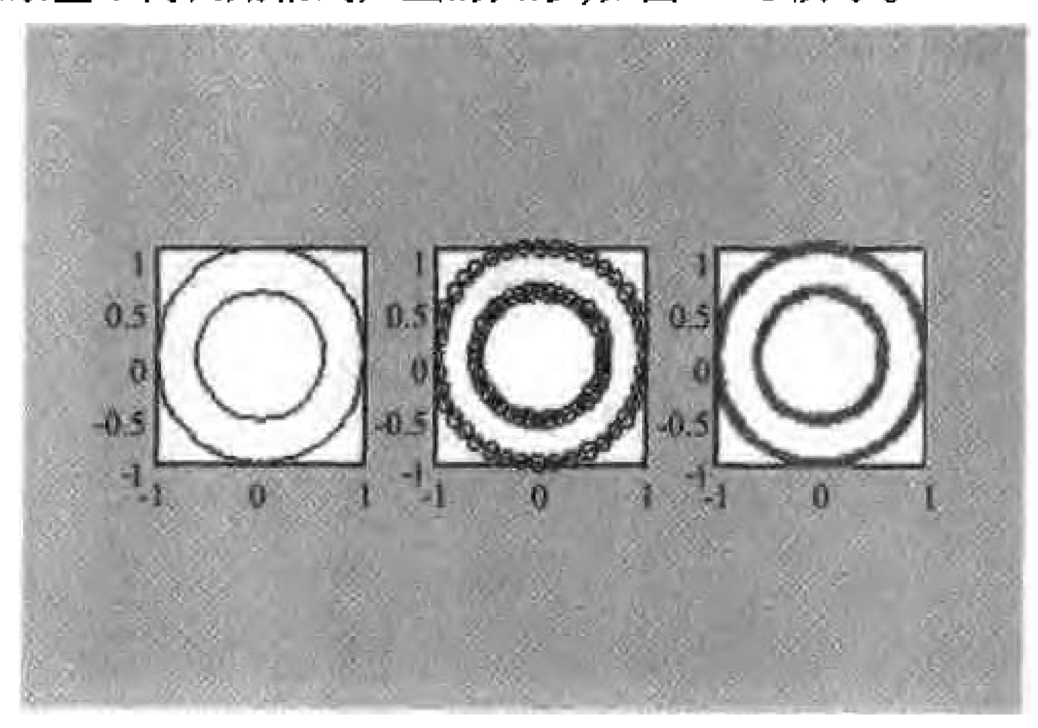


图 2-3　变长度输入宗量不同调用格式产生的图形

2.4 跨空间变量传递

2.4.1 跨空间计算串表达式的值

evalin 运行机理与 eval 的异同。

(1)编写 M 函数文件

```
[evalinzzy.m]
function y1 = evalinzzy(a,s)
t = (0:a)/a * 2 * pi;
y1 = subevalinzzy(4,s);
% ------------- subfunction --------------
function y2 = subevalinzzy(a,s)
t = (0:a)/a * 2 * pi;ss = 'a * exp(i * t)';
switch s
case {'base','caller'}
  y2 = evalin(s,ss);
case 'self'
  y2 = eval(ss);
end
```

(2)在 Notebook 或 MATLAB 指令窗中运行以下指令

```
clear,a = 30;t = (0:a)/a * 2 * pi;sss = {'base','caller','self'};
for k = 1:3
  y0 = evalinzzy(8,sss{k});
  subplot(1,3,k)
  plot(real(y0),imag(y0),'r','LineWidth',3),axis square image
end
```

利用不同工作空间中的变量值计算 eval('a * exp(i * t)')的图形,如图 2-4 所示。

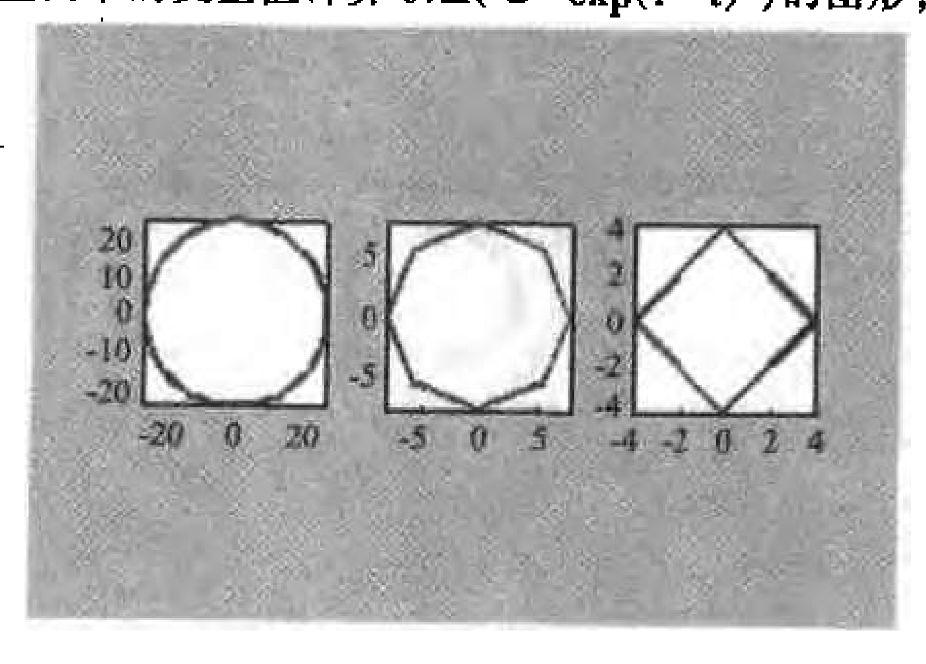


图 2-4 利用不同工作空间中的变量值计算 eval('a * exp(i * t)')

2.4.2 跨空间赋值

assignin 运作机理示范。

(1)编写 M 函数文件

[assigninxyq.m]

```
function y = assigninxyq(x)
y = sqrt(x);t = x^2;
assignin('base','yy',t)
```

(2)在 Notebbok 或 MATLAB 指令窗中运行以下指令

```
clear;x = 4;y = assigninxyq(x);
disp([blanks(5),'x',blanks(5),'y',blanks(4),'yy']),disp([x,y,yy])
     x     y    yy
     4     2    16
```

2.5 串演算函数

2.5.1 eval

计算"表达式"串,产生向量值。

```
clear,t = pi;cem = '[t/2,t * 2,sin(t)]';y = eval(cem)
y =
    1.5708    6.2832    0.0000
```

计算"语句"串,创建变量。

```
clear,t = pi;eval('theta = t/2,y = sin(theta)');who
theta =
  1.5708
y =
    1
Your variables are:
t      theta    y
```

计算"替代"串。

```
A = ones(2,1);B = ones(1,3);c = eval('B * A','A * B'),errmessage = lasterr
c =
     1     1     1
     1     1     1
errmessage =
Error using = = > *
Inner matrix dimensions must agree.
```

计算"合成"串。

```
CEM = {'cos','sin','tan'};
for k = 1:3
    theta = pi * k/12;
    y(1,k) = eval([CEM{1},'(',num2str(theta),')']);
end
y
y =
    0.9659    0.8660    0.7071
```

2.5.2 feval

feval 和 eval 运行区别之一:feval 的 FN 绝对不能是表达式。

```
x = pi/4; Ve = eval('1 + sin(x)')
Ve =
    1.7071

Vf = feval('1 + sin(x)',x)
??? Cannot find function '1 + sin(x)'.
```

feval 和 eval 调用区别:feval 的 FN 只接受函数名。本例两种方法以后者为好。

```
randn('seed',1);A = rand(2,2);
[ue,de,ve] = eval('svd(A)');
disp('Results by eval');disp([ue,de,ve]);disp(blanks(1))
[uf,df,vf] = feval('svd',A);
disp('Results by feval');disp([uf,df,vf])
Results by eval
  0.6180    0.7862    0.5516         0    0.4616    0.8871
  0.7862   -0.6180         0    0.3038    0.8871   -0.4616

Results by feval
  0.6180    0.7862    0.5516         0    0.4616    0.8871
  0.7862   -0.6180         0    0.3038    0.8871   -0.4616
```

2.6 内联函数创建和应用示例

内联函数的第一种创建格式;使内联函数适于"数组运算"。

```
clear, F1 = inline('sin(rho)/rho')        %第一种格式创建内联函数
F1 =
   Inline function:
   F1(rho) = sin(rho)/rho
```

```
f1 = F1(2)                          %内联函数的一种使用方法
f1 =
    0.4546

FF1 = vectorize(F1)                 %产生适于"数组运算"的内联函数
xx = [0.5,1,1.5,2];ff1 = FF1(xx)
FF1 =
   Inline function:
   FF1(rho) = sin(rho)./rho
ff1 =
  0.9589   0.8415   0.6650   0.4546
```

第一种内联函数创建格式的缺陷;含向量的多宗量输入的赋值。

```
G1 = inline('a * exp(x(1)) * cos(x(2))'),G1(2,[-1,pi/3])
G1 =
     Inline function:
     G1(a) = a * exp(x(1)) * cos(x(2))
  Error using ==> inline/subsref
Too many inputs to inline function.

G2 = inline('a * exp(x(1)) * cos(x(2))','a','x'),G2(2,[-1,pi/3])
G2 =
     Inline function:
     G2(a,x) = a * exp(x(1)) * cos(x(2))
ans =
    0.3679
```

产生向量输入、向量输出的内联函数;这种向量函数的调用方法。

```
Y2 = inline('[x(1)^2;3 * x(1) * sin(x(2))]')
argnames(Y2)                        %观察内联函数的输入宗量
Y2 =
   Inline function:
   Y2(x) = [x(1)^2;3 * x(1) * sin(x(2))]
ans =
'x'

x = [4,pi/6];                       %向量输入的赋值
y2 = Y2(x)                          %获得向量输出
y2 =
  16.0000
   6.0000
```

最简练格式创建内联函数;内联函数可被 feval 指令调用。

```
Z2 = inline('P1 * x * sin(x^2 + P2)',2)  %必须是大写字母 P
Z2 =
    Inline function:
    Z2(x,P1,P2) = P1 * x * sin(x^2 + P2)

z2 = Z2(2,2,3)                    %直接计算内联函数
fz2 = feval(Z2,2,2,3)             %注意:这里,应写 Z2,不能写成 'Z2'。
z2 =
   2.6279
fz2 =
   2.6279
```

2.7 调试器应用示例

本例的目标:对于任意随机向量,画出鲜明标志该随机向量均值、标准差的频数直方图(如图 2-4 所示),或给出绘制这种图形的数据。

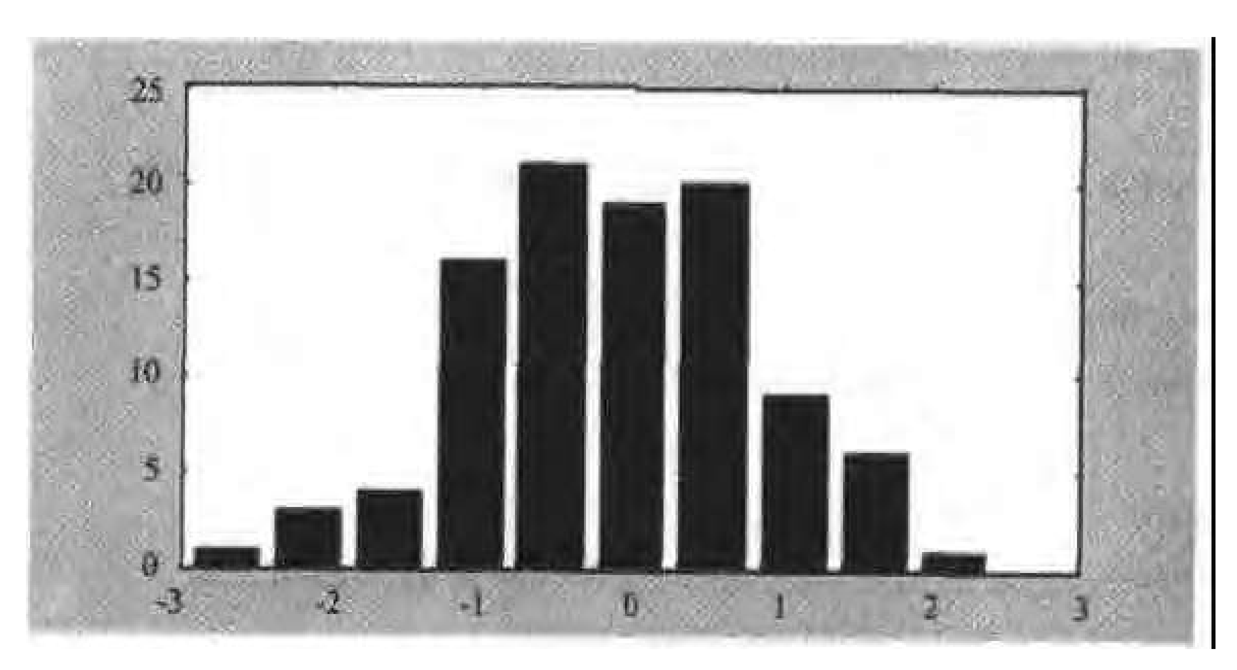


图 2-5　带均值、标准差标志的频数直方图

(1)根据题目要求写出以下两个 M 文件

```
[histzzy.m]
function [nn,xx,xmu,xstd] = histzzy(x)
xmu = mean(x);
xstd = std(x);
[nn,xx] = hist(x);
if nargout = = 0
    barzzy(nn,xx,xmu,xstd)                                    % <6>
end
```

```
[barzzy.m]
function barzzy(nn,xx,xmu,xstd)
clf,
bar(xx,nn);hold on
Ylimit = get(gca,'YLim');
yy = 0:Ylimit(2);
xxmu = xmu * size(yy);
xxL = xxmu/xmu * (xmu - xstd);
xxR = xxmu/xmu * (xmu + xstd);
plot(xxmu,yy,'r','Linewidth',3)
plot(xxL,yy,'rx','MarkerSize',8)
plot(xxR,yy,'rx','MarkerSize',8),hold off
```

(2)初次运行以下指令后，得到运行出错的提示

```
randn('seed',1),x = randn(1,100);histzzy(x);
??? Error using = = > plot
Vectors must be the same lengths.
Error in = = > E: \ mat53 \ work \ barzzy.m
On line 8    = = > plot(xxmu,yy,'r','Linewidth',3)
Error in = = > E: \ MAT53 \ work \ histzzy.m
On line 6    = = >   barzzy(nn,xx,xmu,xstd)
```

运行出错时所得的不完整图形如图 2-5 所示。

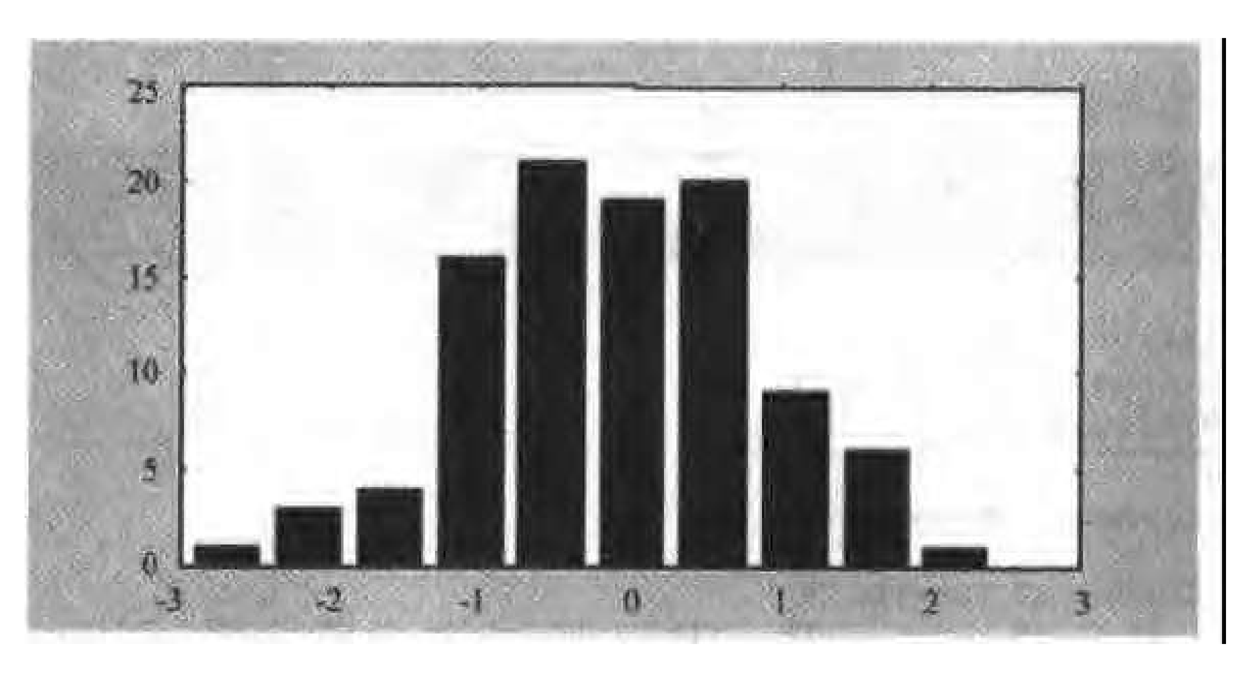


图 2-5　运行出错时所得的不完整图形

(3)初步分析错误原因

(4)断点设置(如图 2-6 所示)

(5)在指令窗中运行以下指令，进入“动态”调试

```
randn('seed',1),x = randn(1,100);histzzy(x);
```

(6)深入被调文件内部(如图 2-7 所示)

(7)连续执行，直到另一个断点

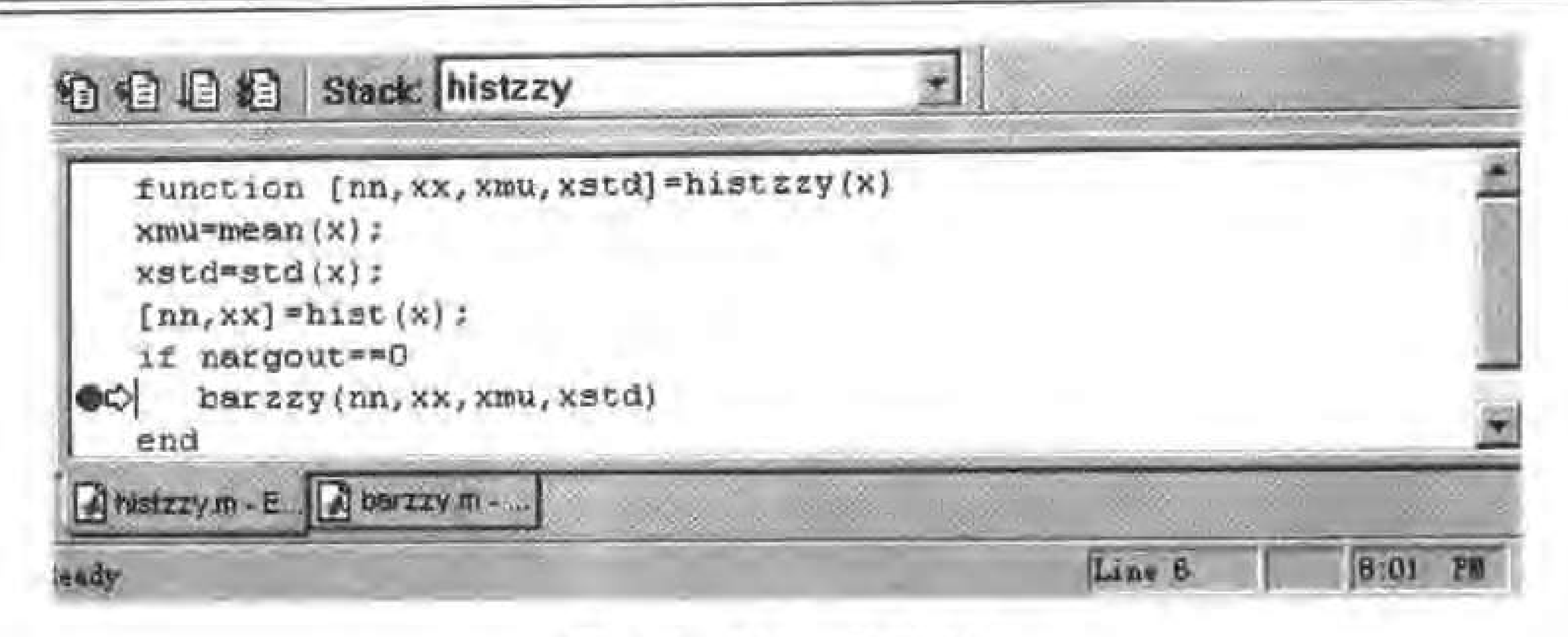


图 2-6 运行暂停在断点处

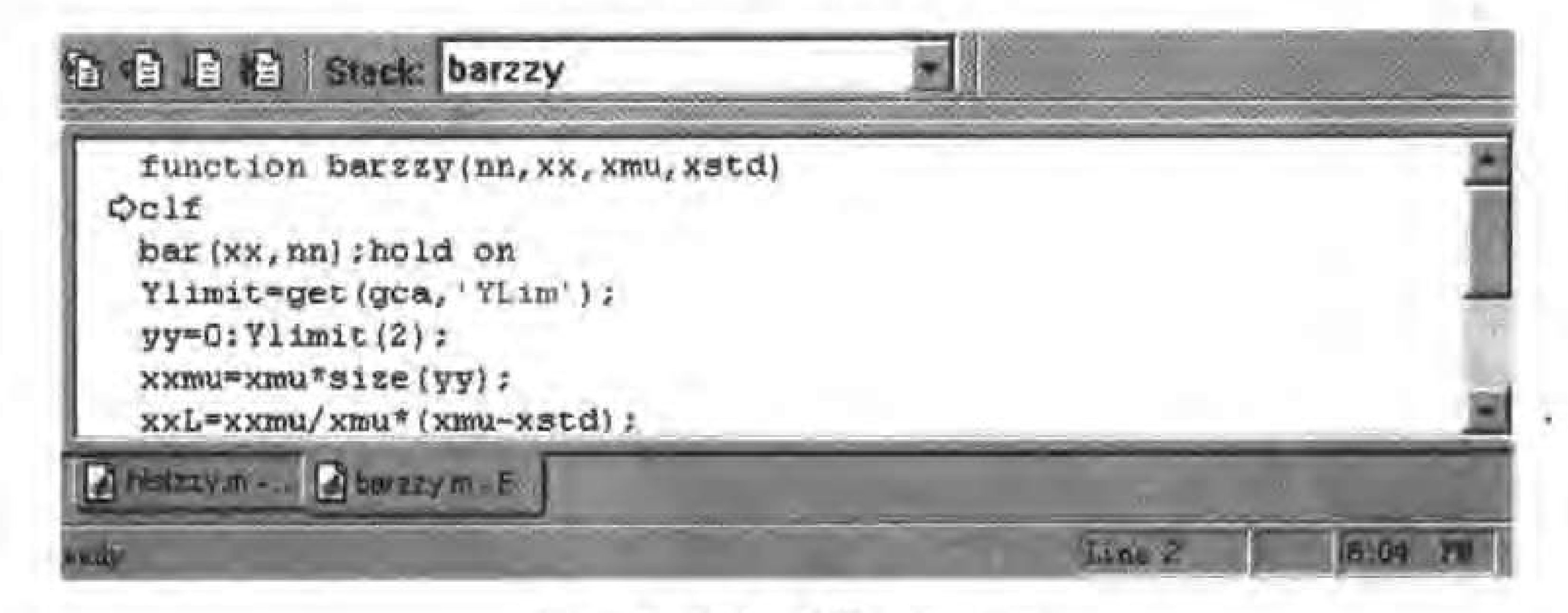


图 2-7 动态调试深入 barzzy.m

(8)指令窗观察法(如图 2-8 所示)

```
Stack: barzzy
clf
bar(xx,nn);hold on
Ylimit=get(gca,'YLim');
yy=0:Ylimit(2);
yy =
 Columns 1 through 12
    0    1    2    3    4    5    6    7    8    9   10   11
 Columns 13 through 24
   12   13   14   15   16   17   18   19   20   21   22   23
 Columns 25 through 31
   24   25   26   27   28   29   30
Line 5
```

图 2-8 变量值的鼠标观察法

(9)修改程序,停止第一论调试,重新运行

randn('seed',1),x = randn(1,100);histzzy(x);

2.8　M 文件性能分析

2.8.1　分析器

对如图 2－4 中那通过调试后运行正确的 histzzy.m 进行性能分析(如图 2－9 所示)。

(1)产生分析报告

```
randn('seed',1),x = randn(1,100);
profile on                        %启动分析器,在 mmex 层面上统计数据
for kk = 1:100;histzzy(x);end     %运行 100 次,使统计数据受随机因素影响较小
profile report histzzy_report     %产生名为 histzzy_report.htm 的分析报告
```

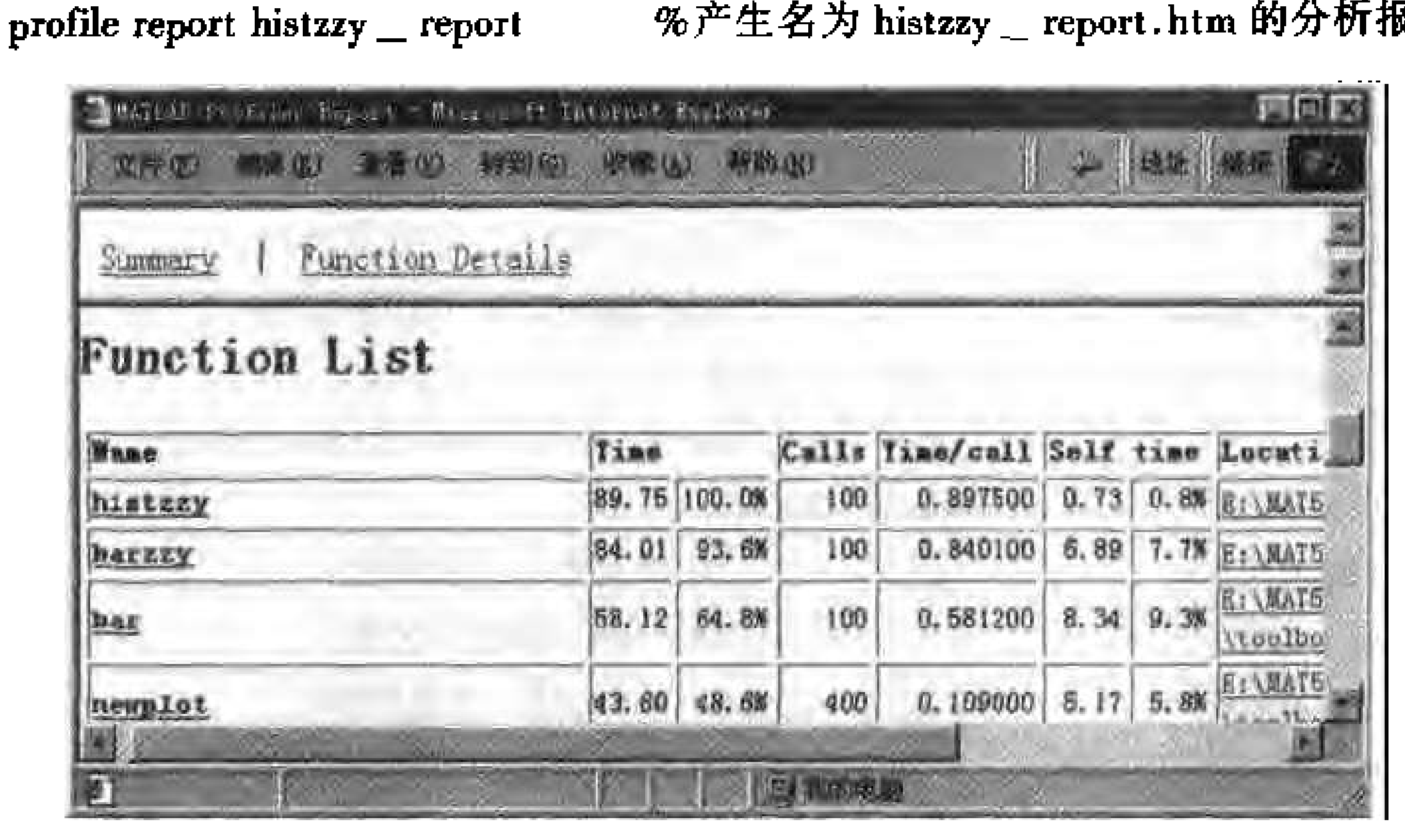


图 2－9　超文本形式分析报告的 Summary 章的部分内容

(2)研究分析报告(如图 2－10 所示)

```
               1:  function barzzy(nn,xx,xmu,xstd)
  8.36 10%     2:  clf
 60.33 72%     3:  bar(xx,nn);hold on
  5.52  7%     4:  Ylimit=get(gca,'YLim');
               5:  yy=0:Ylimit(2);

               8:  xxR=xxmu/xmu*(xmu+xstd);
  2.81  3%     9:  plot(xxmu,yy,'r','Linewidth',3)
  2.61  3%    10:  plot(xxL,yy,'rx','MarkerSize',8)
  2.78  3%    11:  plot(xxR,yy,'rx','MarkerSize',8)
  1.39  2%    12:  hold off
```

图 2－10　从所得 barzzy.m 明细报告中剪辑的部分内容

2.9 面向对象编程

2.9.1 面向对象编程应用示例

创建"先进先出"FIFO 队列 queue 类的全过程。在本例中,读者应充分注意:构架域(Fields of a structure array)和定义在其上的方法函数(Method function)之间的关系。

(1)建立类目录 @queue

(2)选择构架数组为 queue 类的数据结构

(3)创建构造函数 queue.m

```
[@queue \ queue.m]
function q = queue(v)
%@QUEUE \ QUEUE   queue class constructor function
% 调用格式
%    q = queue                     创建一个"空"队列对象
%    q = queue(v)                  创建包含变量 v 的队列对象
superiorto('double','sparse','struct','cell','char','inline','sym');
                                   % 使 queue 对象具有最高优先级    <6>
if nargin > 1;error('Too many arguments.');end;
if nargin = = 0                    % 没有输入宗量情况
   q.value = [  ];                 % value 域被赋"空阵"
   q.name = '';                    % name 域不给任何字符
   q = class(q,'queue');           % 给变量 q 挂上 queue 标签
elseif isa(v,'queue');             % 输入宗量是同类对象情况
   q = v;                          % 直接把输入量赋给 q
else                               % 非同类输入宗量情况
   q.value = v;                    % 在 value 域中放置输入对象 v 的内容
   q.name = inputname(1);          % 在 name 域中放置输入对象名 v 字符
   if isempty(q.name)              % 假如输入量无名
     q.name = ['(' class(v) ')'];  % 就采用 v 本身的类名
   end
   q = class(q,'queue');           % 给变量 q 挂上 queue 标签         <20>
end
```

(4)为 queue 对象编写显示函数 display.m

```
[@queue \ display.m]
function display(q,ki,kj)
%QUEUE \ DISPLAY   command window display of a queue object.
% 调用格式
```

```
%    display(q)                      笼统显示整个队列
%    display(q,ki)                   单下标法显示具体队列元素的内容
%    display(q,ki,kj)                双下标法显示具体队列元素的内容
if nargin = = 0;error('缺少输入宗量,即被显示对象!');end
switch nargin
case 1                                  % 显示整个队列
   [m,n] = size(q);
vname = inputname(1);                   % 被显示对象 q 的名称
if isempty(vname)                       % 显示对象若无名称
   fprintf('ans = \n');                 % 按 MATLAB 惯例,屏幕显示 ans 缺省名
elseif fprintf('%s = \n',vname);        % 对象有名称时,则屏幕以字符串形式显示名称
end;
if isempty(q)                           % 假如被显示对象为"空"
fprintf('    [ empty   ')                                                    % <17>
   fprintf('%s',class(q))                                                    % <18>
  fprintf(' ] \n \n');                                                       % <19>
  elseif m * n = = 1;                   % 被显示对象今包含一个"元素"时
   fprintf('    %s: ',q.name);          % 屏幕先以字符串形式显示所存放对象的名称
   disp(q.value);                       % 紧接着,不换行,显示所存放对象的内容
   fprintf(' \n');
else                                    % 被显示对象今包含多个"元素"时
fprintf('    [ %d * %d ',m,n)           % 以下 3 条指令只显示队列"元素"排列 <25>
fprintf('%s',class(q))                                                       % <26>
  fprintf(' ] \n \n');                                                       % <27>
  end
case 2                                  % 单下标法显示具体队列元素的内容
   disp(['The content of ',inputname(1),'(',int2str(ki),')'])
   disp(['is a ',class(q(ki).value),'object'])
   fprintf('    %s = \n',q(ki).name);
   disp(q(ki).value);
   fprintf(' \n');
case 3                                  % 双下标法显示具体队列元素的内容
   disp(['The content of ',inputname(1),'(',int2str(ki),',',int2str(kj),')'])
   disp(['is a ',class(q(ki,kj).value),' object'])
   fprintf('    %s = \n',q(ki,kj).name);
   disp(q(ki,kj).value);
   fprintf(' \n');
end
```

(5)编写重载函数 isempty.m。

```
[@queue \ isempty.m]
function f = isempty(q)
% @QUEUE \ ISEMPTY    True for an empty queue object.
f = 0;
[m,n] = size(q);
if m * n = = 1;
  if isempty(q.value) & isempty(q.name)
    f = 1;
  end;
end;
```

(6)编写“入队”、“离队”函数

```
[ @queue \ comein.m]
function q = comein(p,varargin)
% @QUEUE \ COMEIN    a variable comes to the end of a queue.
% 调用格式
%   comein(p,a,b,…)              使输入宗量 a,b 等排在 p 之后形成新队列,
%                                 其名沿用 p 位置上的输入队列名
%   q = comein(p,a,b,...)        使输入宗量 a,b 等排在 p 之后形成新队列 q
if nargin < 2 error('comein needs at least two arguments.');end;
if isa(p,'queue') error([inputname(1),' is not a queue']);end;
q0 = p;
qzzy = class(p);               %获取第一输入宗量的类别字符串        <10>
for i = 1:length(varargin)
   temp = varargin{i};
   s = eval([qzzy,'(temp)']);   %使后来元素成为与第一输入宗量相同的类别
                                                                   <13>
   s.name = inputname(i + 1);
   if isempty(s.name)          %假如某输入宗量本身无名称
s.name = ['(' class(temp) ')'];   %则把它的类别名作为名称使用
   end
   if isempty(q0)              %假如前队列是“空”队列
     q0 = s;                   %则直接进入队列
   else                        %假如前队列非“空”
     q0 = [q0 s];              %则新变量排在队尾
   end
end
if nargout = = 0;              %假如没有输出宗量
  assignin('caller',inputname(1),q0); %新队列沿用第一个输入队列名
  evalin('caller',inputname(1));
```

```
else                                  %假如有输入输出宗量
   q = q0;                            %新队列名为 q
end
[@queue \ goout.m]
function [n,v,q] = goout(p)
% @QUEUE \ GOOUT  removes the first(the front) element from a queue.
% 调用格式
%  goout(p)                          从队列 p 中的第一个元素离队
%  v = goout(p)                      v 是从 p 队列中移出的那第一个元素的“值”
%  [n,v] = goout(p)                  n,v 分别是从 p 队列中移出的那第一个元素的
                                     “名称”和“值”
%  [n,v,q] = goout(p)                n,v 分别是从 p 队列中移出的那第一个元素的
                                     “名称”和“值”
%                                    q 是被移去第一个元素后的新队列
if nargin = = 0 ;error('No queue specifide.');end;
if nargout > 3;error('Too many output arguments.');end;
if nargin > 1 error('Too many input arguments.');end;
if isa(p,'queue');error([inputname(1),' is not a queue.']);end;
if isempty(p)
   q1 = p;
else
   [m,n] = size(p);
   v1 = p(1).value;n1 = p(1).name;
   if m * n = = 1
     q1 = queue;
   else
     q1 = p(2:end);
   end
end
if nargout < 3;
   assignin('caller',inputname(1),q1);
end;
if nargout = = 0,
   evalin('caller',inputname(1));
end
if nargout > = 1;v = v1;end;
if nargout > = 2;n = n1;end;
if nargout = = 3;q = q1;end;
```

本例的目的：一，检验上例所编写的程序的正确性；二，演示所设计的新类是如何被运

作的。

(1)创建一个队列对象,并显示

```
qe = 'Hello!    你好 !';                %字符串
Q = queue(qe)                           %生成队列对象 Q,并显示
Q =
   qe: Hello!       你好!
```

(2)类别检查和是否对象判断

```
class(Q)                                %类别检查
isobject(Q)                             %是否对象判断
isa(Q,'queue')                          %是否队列判断
ans =
queue
ans =
     1
ans =
     1
```

(3)是否"空"队列判断

```
isempty(Q)
ans =
     0
```

一,演示"入队"、"离队"函数的调用方法;二,演示@queue \ display 显示队列具体元素细节的功能。

(1)利用"入队"函数,使队列变长

```
a=[1,2,3;4,5,6];b{1} = 'This';b{2} = ' is ';b{3} = 'a cell array';
comein(Q,a,b)    %增长队列,并显示整个队列的"宏观"情况
Q =
   [ 1 * 3 queue ]
```

(2)显示队列 Q 中具体元素的内容

```
display(Q,2)                       %给出 Q 队列第 2 个元素的类别、内容:变量名和值
The content of Q(2)
is a 'double' object
  a=
    1     2     3
    4     5     6
```

(3)把 Q 队列第 1 个元素和其余部分分离,并生成新队列 QQ

```
[nn,vv,QQ] = goout(Q)
nn =
qe
vv =
```

```
Hello!    你好!
QQ =
    [ 1 * 2 queue ]
```

(4)采用双下标法,显示 QQ(1,2)的内容

```
display(QQ,1,2)
The content of QQ(1,2)
is a 'cell' object
  b =
  'This'   'is'   'a cell array'
```

利用指令 methods 可以获知对任何类定义的(在类目录上的)所有方法函数。

```
methods queue
Methods for class queue:

comein    display    goout    isempty    queue
```

2.10　继承性创建子类的示例

把上节的第一个例子构成的队列作为父类,利用继承性,创建 stack 堆栈子类。

(1)建立类目录 @stack ,并使它成为当前目录(以下指令仅对 5.3 版适用)

```
mkdir('e: \ mat53 \ work','@stack')    %在 e: \ mat53 \ work 上建子目录@stack
cd e: \ mat53 \ work \ @stack %是子目录@stack 成为当前目录
```

(2)编写堆栈类的构造函数@stack \ stack.m。

```
[@stack \ stack.m]
function ST = stack(v)
% 调用格式
%     ST = stack                   创建一个"空"堆栈对象
%     ST = stack(v)                创建包含变量 v 的堆栈对象
if nargin > 1;error('Too many arguments.');end;
if nargin = = 0                    % 没有输入宗量情况
   Q = queue;
   s.value = [];                   % value 域被赋"空阵"
   s.name = '';                    % name 域不给任何字符
elseif isa(v,'stack');             % 输入宗量是同类对象情况
   s = v;                          % 直接把输入量赋给 q
   Q = queue(evalin('caller',inputname(1)));  %生成队列对象
else                               % 非同类输入宗量情况
   s.value = v;                    % 在 value 域中放置输入对象 v 的内容
   s.name = inputname(1);          % 在 name 域中放置输入对象名 v 字符
```

```
    if isempty(s.name)              % 假如输入量无名
  s.name = ['(' class(v) ')'];      % 就采用 v 本身的类名
    end
    Q = queue(evalin('caller',inputname(1)));   %生成队列对象
  end
  ST = class(s,'stack',Q);          % 产生继承父类对象 Q 性质的 ST 堆栈子类
```

检查上例构造函数设计的正确性,并观察堆栈关于队列的显示,类别判断和为"空"判断性质的继承。

(1)堆栈对象的创建,及由于继承性,而获得正确显示

```
AA = '继承性';
ST = stack(AA)
ST =
    AA:  继承性
```

(2)类别检查

```
class(ST)
ans =
stack
```

(3)由于在此设计采用了继承性,所以"是堆栈,就一定是队列"

```
isa(ST,'stack')
isa(ST,'queue')
ans =
    1
ans =
    1
```

(4)检查对 isempty.m 是被继承

```
isempty(ST)
ans =
    0
```

本例通过堆栈类对象的"压入"和"弹出"操作,进一步观察继承性。

(1)把元素压入堆栈

```
BB = 1:6;CC = sym('x^2 + 4 * x');
comein(ST,BB,CC)
ST =
    [ 1 * 3 stack ]
```

(2)显示堆栈中第三元素的内容

```
display(ST,3)
The content of ST(3)
is a 'sym' object
  CC =
```

```
x^2 + 4 * x
```

(3)从堆栈弹出元素

```
[Name1, Value1, ST_1] = goout(ST)
Name1 =
AA
Value1 =
  继承性
ST_1 =
    [ 1 * 2 stack ]
```

第3章　图形用户界面的制作

用户界面(或接口)是指:人与机器(或程序)之间交互作用的工具和方法。如键盘、鼠标、跟踪球、话筒都可成为与计算机交换信息的接口。

图形用户界面(Graphical User Interfaces, GUI)则是由窗口、光标、按键、菜单、文字说明等对象(Objects)构成的一个用户界面。用户通过一定的方法(如鼠标或键盘)选择、激活这些图形对象,使计算机产生某种动作或变化,比如实现计算、绘图等。

假如读者所从事的数据分析、解方程、计算结果可视工作比较单一,那么一般不会考虑 GUI 的制作。但是如果读者想向别人提供应用程序,想进行某种技术、方法的演示,想制作一个供反复使用且操作简单的专用工具,那么图形用户界面也许是最好的选择之一。

MATLAB 为表现其基本功能面设计的演示程序 demo 是使用图形界面的最好范例。MATLAB 的用户,在指令窗中运行 demo 打开那图形界面后,只要用鼠标进行选择和点击,就可浏览那丰富多彩的内容。

即便比较熟悉 MATLAB 的读者,在他初次编写 GUI 程序时,也会感到棘手。为使读者获得制作自己 GUI 的体验,本章 3.1 节提供了一个简单的示例。读者只要输入所提供的程序,就可引出相应的界面。

本章 3.2 节叙述图形用户界面的设计原则和一般制作步骤。第 3.3、3.4 节分别介绍用户菜单、用户控件的制作。出于“由浅入深”的考虑,前 4 节制作 GUI 是通过 M 脚本文件实现的。利用 M 函数文件制作 GUI,需要解决数据传递问题,为此专设 3.5 节给予阐述和示例。MATLAB 5.x 为方便用户制作图形界面,提供了一个交互式的设计工具 guide。关于该工具的使用方法,被放在 3.6 节中,以一个综合例题为设计目标逐步展开。

在此提醒读者,假如要比较准确地理解本章程序和掌握本章内容,请先阅读第 10 章关于图柄的内容。

3.1　入　　门

对于传递函数为 $G=\frac{1}{s^2+2\zeta s+1}$ 的归一化二阶系统,制作一个能绘制该系统单位阶跃响应的图形用户界面。本例演示以下内容:(A)图形界面的大致生成过程;(B)静态文本和编辑框的生成;(C)坐标方格控制键的形成;(D)如何使用该界面。

(1)产生图形窗和轴位框

```
clf reset
```

```
H = axes('unit','normalized','position',[0,0,1,1],'visible','off');
set(gcf,'currentaxes',H);
str = '\fontname{隶书}归一化二阶系统的阶跃响应曲线';
text(0.12,0.93,str,'fontsize',13);
h_fig = get(H,'parent');
set(h_fig,'unit','normalized','position',[0.1,0.2,0.7,0.4]);
h_axes = axes('parent',h_fig,...
    'unit','normalized','position',[0.1,0.15,0.55,0.7],...
    'xlim',[0 15],'ylim',[0 1.8],'fontsize',8);
```

产生坐标轴如图 3－1 所示。

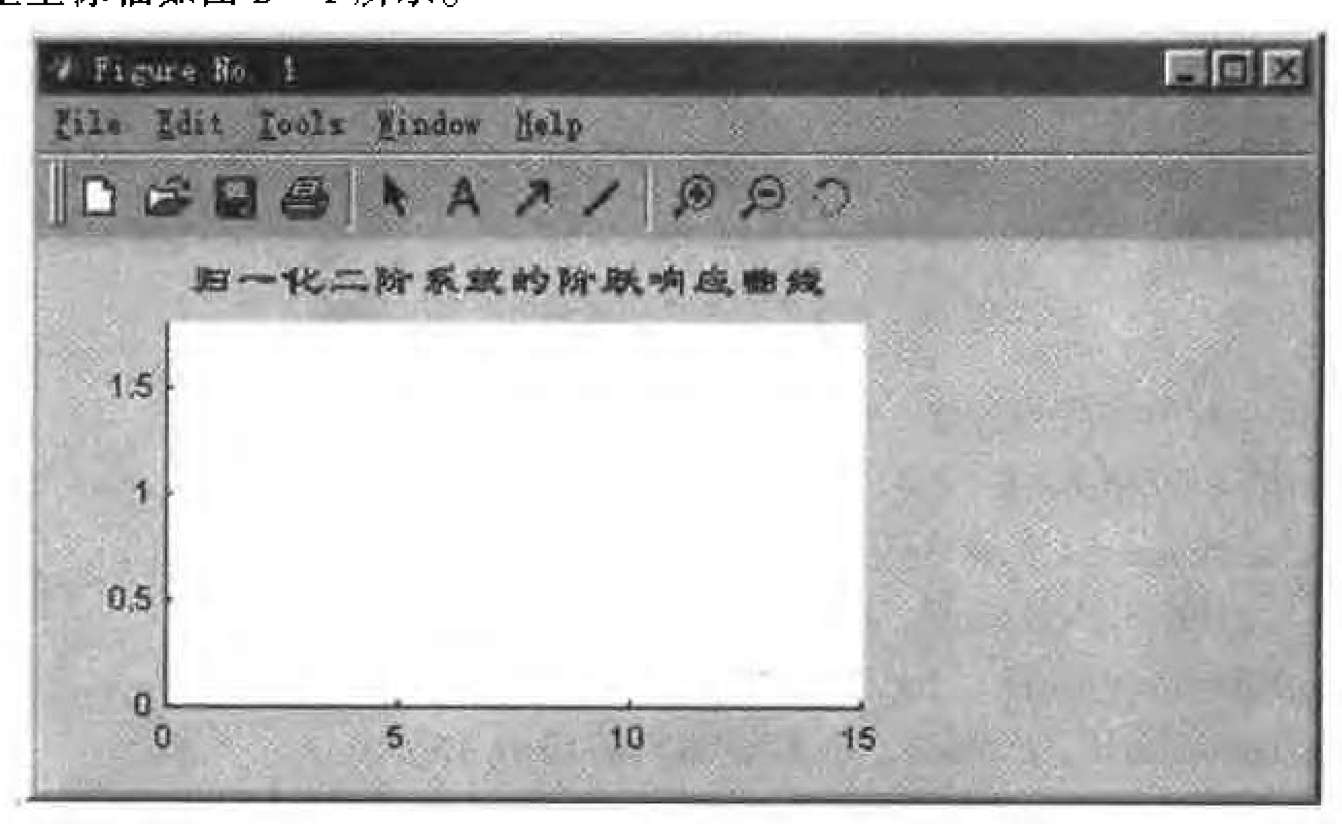


图 3－1　产生坐标轴

(2)在坐标框右侧生成作解释用的“静态文本”和可接受输入的“编辑框”

```
h_text = uicontrol(h_fig,'style','text',...
'unit','normalized','position',[0.67,0.73,0.25,0.14],...
'horizontal','left','string',{'输入阻尼比系数','zeta ='});
h_edit = uicontrol(h_fig,'style','edit',...
'unit','normalized','position',[0.67,0.59,0.25,0.14],...
'horizontal','left',...
'callback',[...
'z = str2num(get(gcbo,''string''));',...
't = 0:0.1:15;',...
'for k = 1:length(z);',...
's2 = tf(1,[1 2*z(k) 1]); ',...
'y(:,k) = step(s2,t);',...
'plot(t,y(:,k));',...
'if (length(z) > 1) ,hold on,end,',...
```

```
'end;',…
'hold off,']);
```

在图形界面中添加编辑框和文本框如图 3-2 所示。

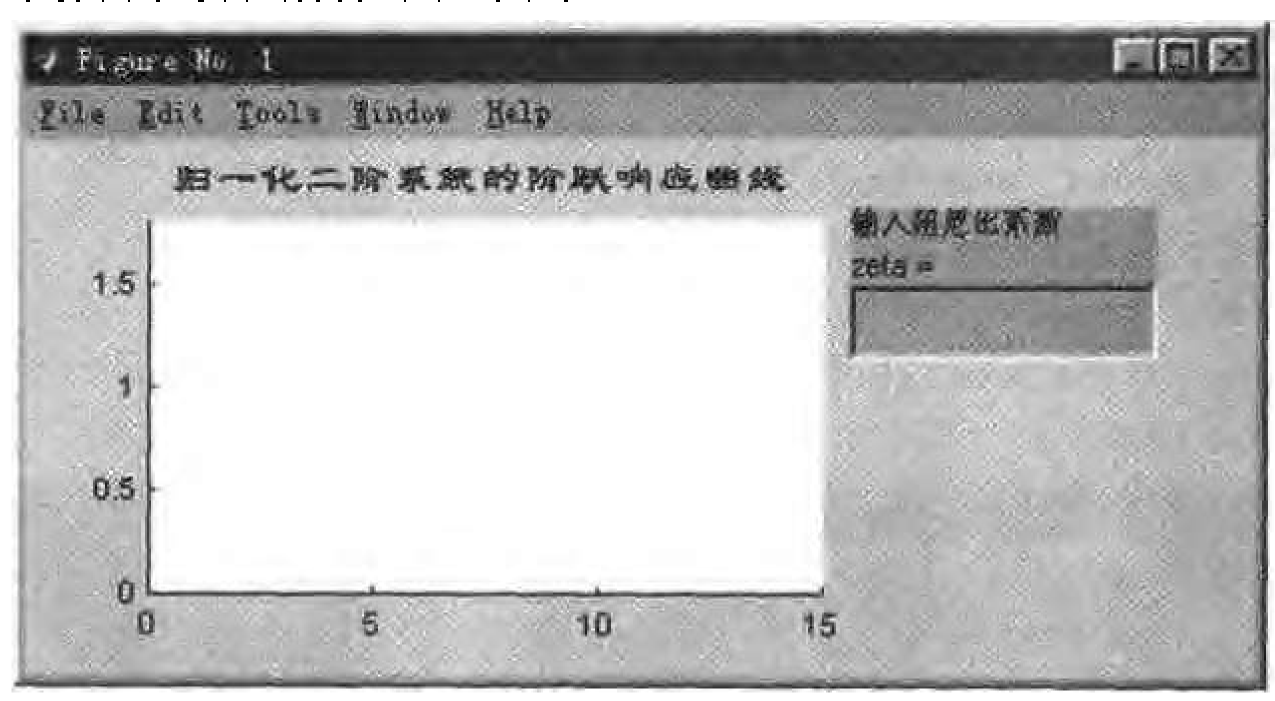


图 3-2 在图形界面中添加编辑框和文本框

(3)形成坐标方格控制按键

```
h_push1 = uicontrol(h_fig,'style','push',…
'unit','normalized','position',[0.67,0.37,0.12,0.15],…
'string','grid on','callback','grid on');
h_push2 = uicontrol(h_fig,'style','push',…
'unit','normalized','position',[0.67,0.15,0.12,0.15],…
'string','grid off','callback','grid off');
```

图 3-3 表示添加了两个按键的图形界面。

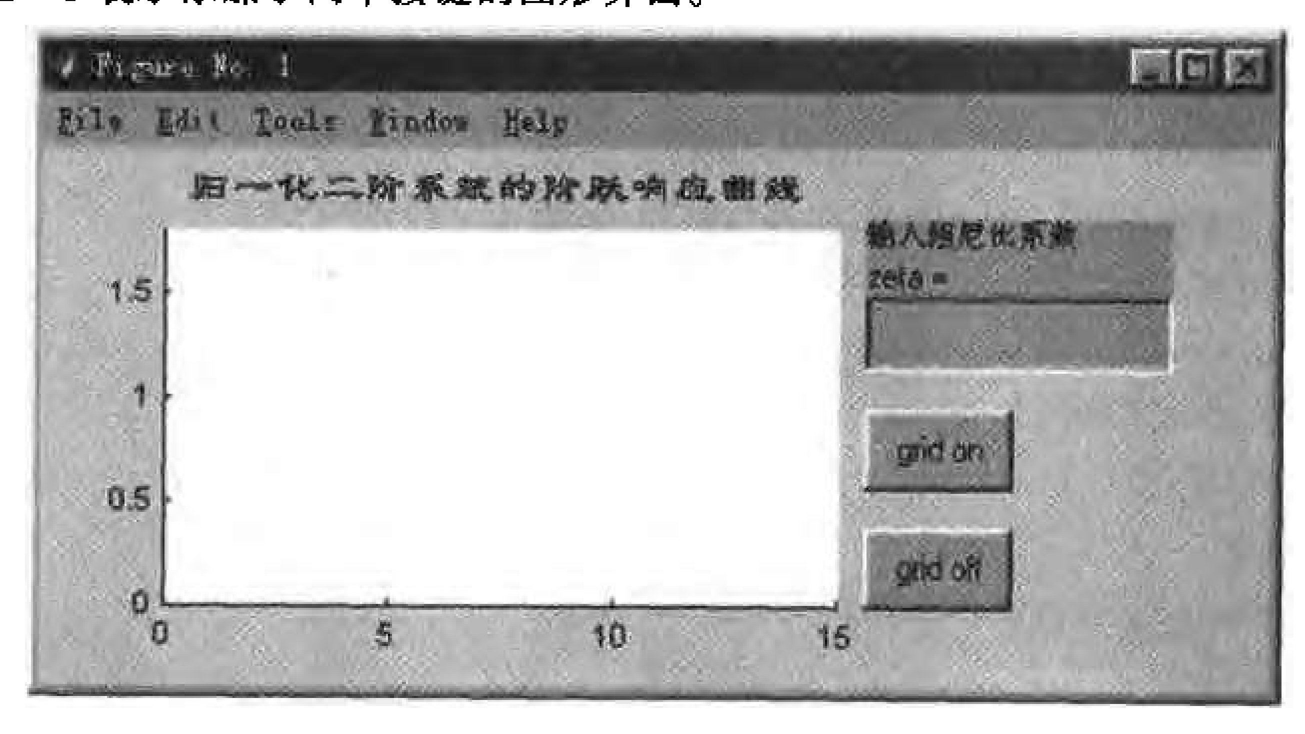


图 3-3 添加了两个按键的图形界面

(4)输入阻尼比系数 ζ,可得单位阶跃响应曲线

输入阻尼比和阻尼比数组所得的响应曲线,分别如图 3-4 和图 3-5 所示。

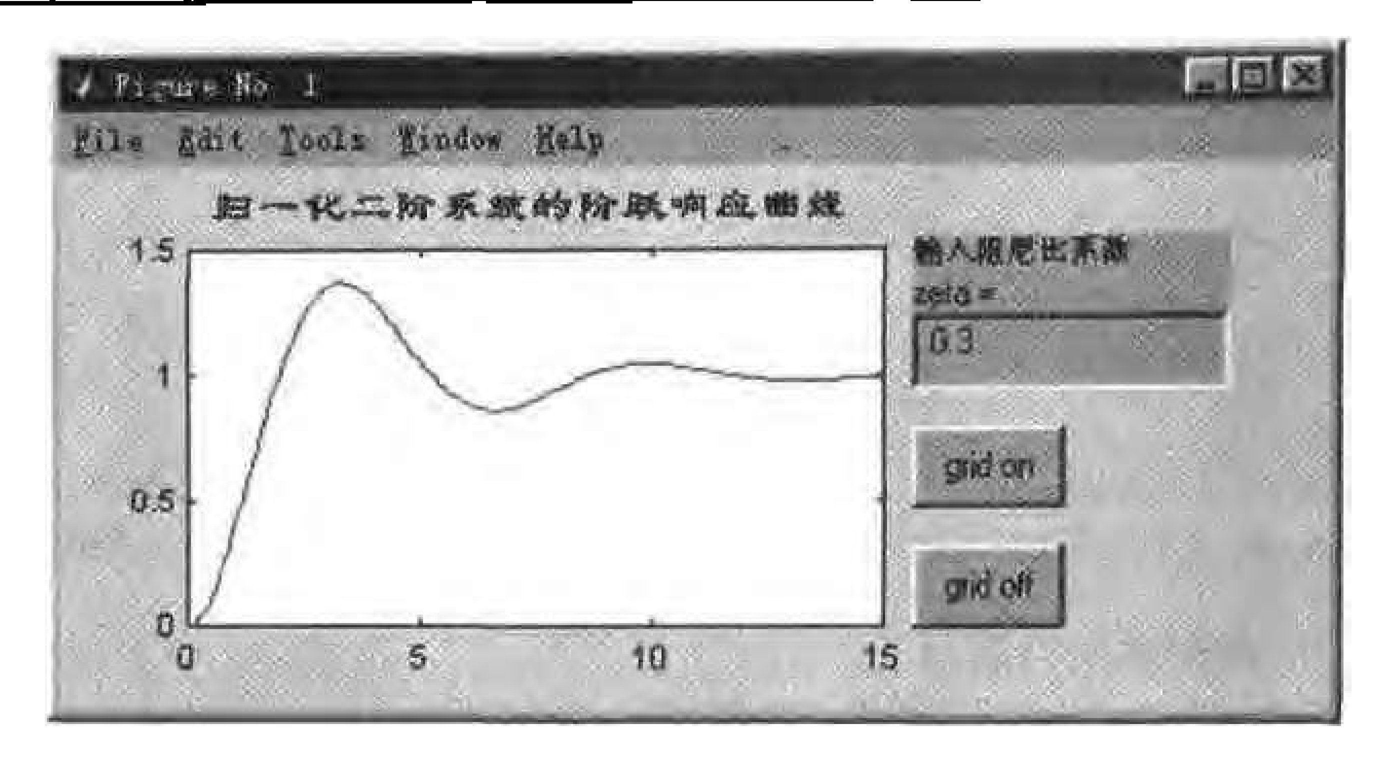


图 3-4 输入阻尼比所得到的响应曲线

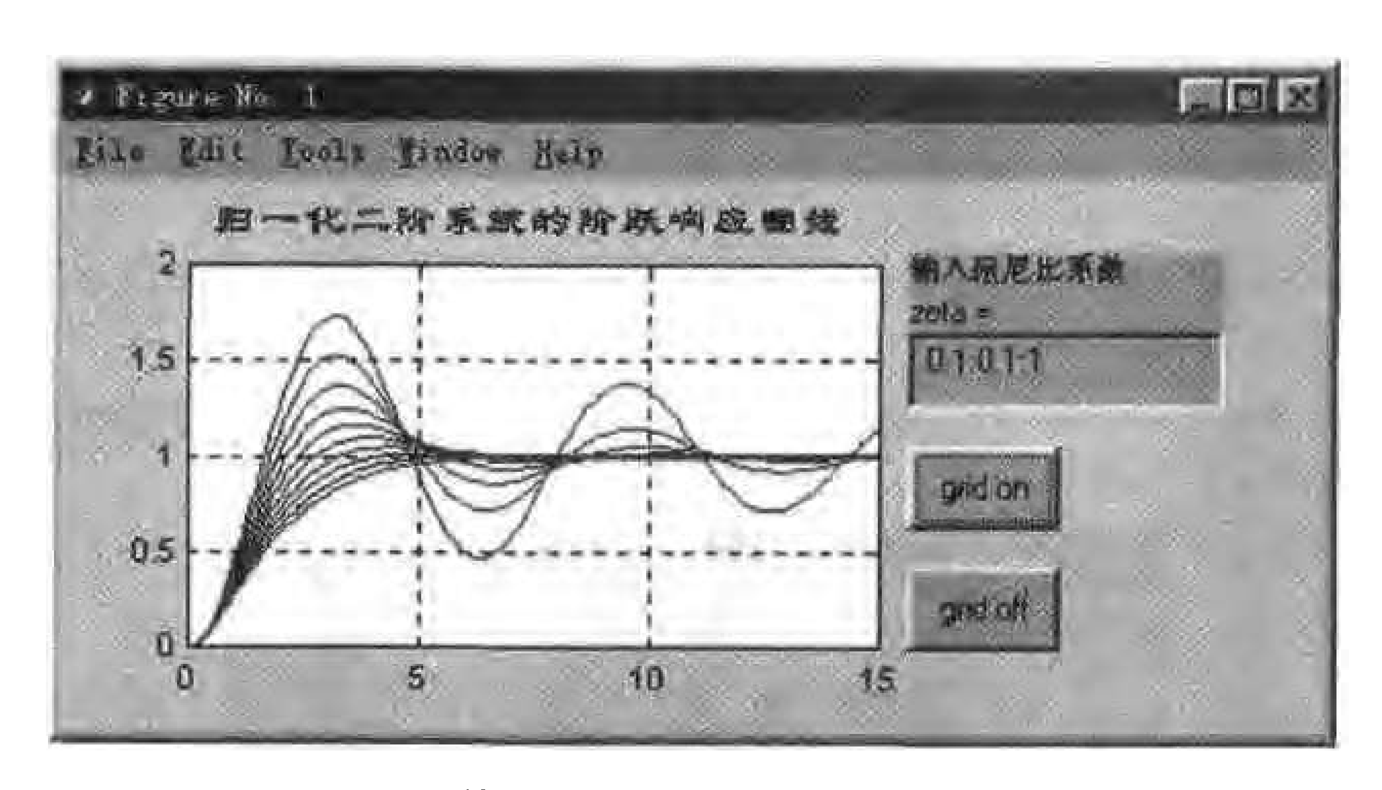


图 3-5 输入阻尼比数组所得到的一组响应曲线

3.2 界面菜单

3.2.1 图形窗的标准菜单

如何隐藏和恢复标准菜单的显示。

(1)获得缺省设置的标准菜单

figure

(2)隐去标准菜单的两种方法

set(H _ fig , 'MenuBar','none');

set(gcf,'menubar',menubar);

(3)恢复图形窗上标准菜单(如图 3-6 和图 3-7 所示)

set(gcf,'menubar','figure');

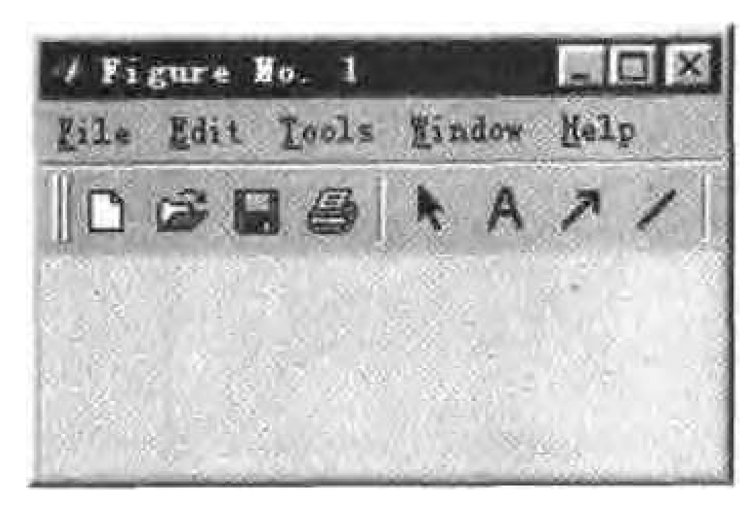



图 3-6 含有菜单条的图形窗

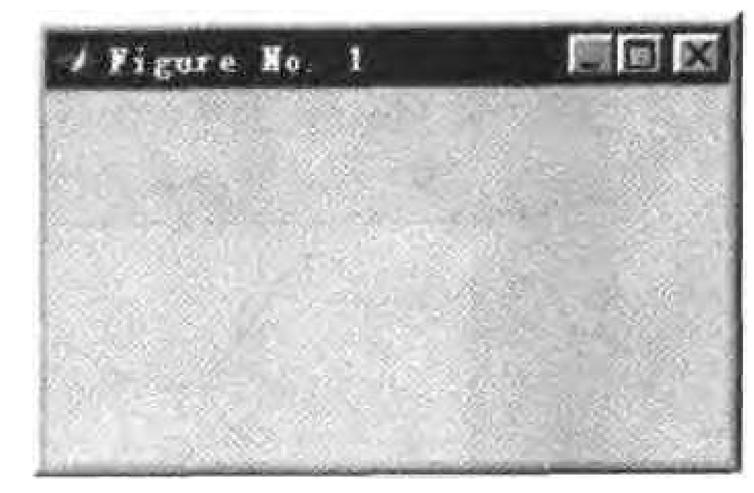


图 3-7 移去菜单条的图形窗

3.2.2 自制的用户菜单

如何自制一个带下拉菜单表的用户菜单(如图 3-8 所示)。该菜单能使图形窗背景颜色设置为蓝色或红色。在命令窗口输入如下命令。

```
figure                                                    %创建一个图形窗
h_menu = uimenu(gcf,'label','Color');                     %制作用户顶层菜单项 Color
                                                                              <2>
h_submenu1 = uimenu(h_menu,'label','Blue',…   %制作下拉菜单项 Blue      <3>
'callback','set(gcf,''Color'',''blue'')');                                 %<4>
h_submenu2 = uimenu(h_menu,'label','Red',…    %制作下拉菜单 Red         <5>
     'callback','set(gcf,''Color'',''red'')');                             %<6>
```



图 3-8 创建用户菜单示例

3.2.3 用户菜单的属性

1. 回调属性和菜单名

在图形窗上自制一个名为【Test】的“顶层菜单项”;当用鼠标点动该菜单项时,将产生一个带分格的封闭坐标轴。通过本例说明:(A)回调属性的运作机理;(B)用户顶层菜单项的制作(C)uimenu 属性的设置方法;(D)复杂字符串的构成方法和注意事项。

(1)在 MATLAB 指令窗中运行以下程序可产生带分格的封闭坐标轴(如图 3-9 所示)

```
grid on,set(gca,'box','on')
```

(2)在 MATLAB 指令窗中用以下 eval 指令可产生与图 3-9 相同的界面

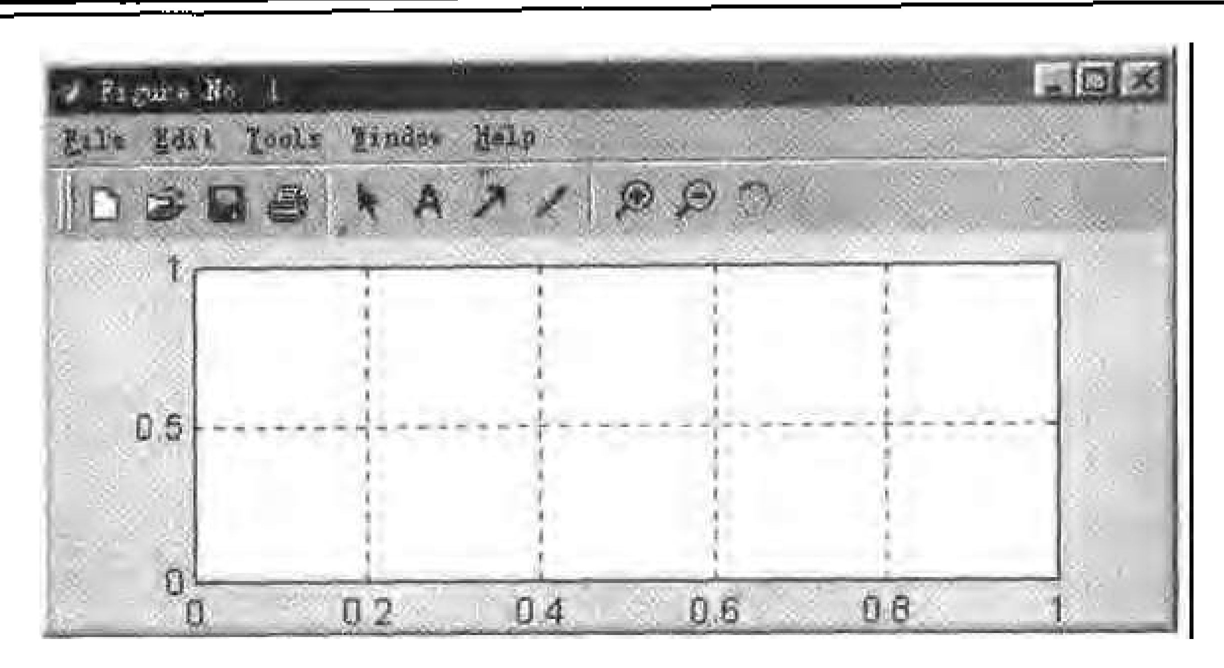


图 3-9　带分格的封闭坐标轴

```
eval('grid on,set(gca,''box'',''on'')')
```

(3)产生图 3-10 界面的 uimenu 的书写格式之一:直接连续表示法

```
uimenu('Label','Test','Callback','grid on,set(gca,''box'',''on''),')
```

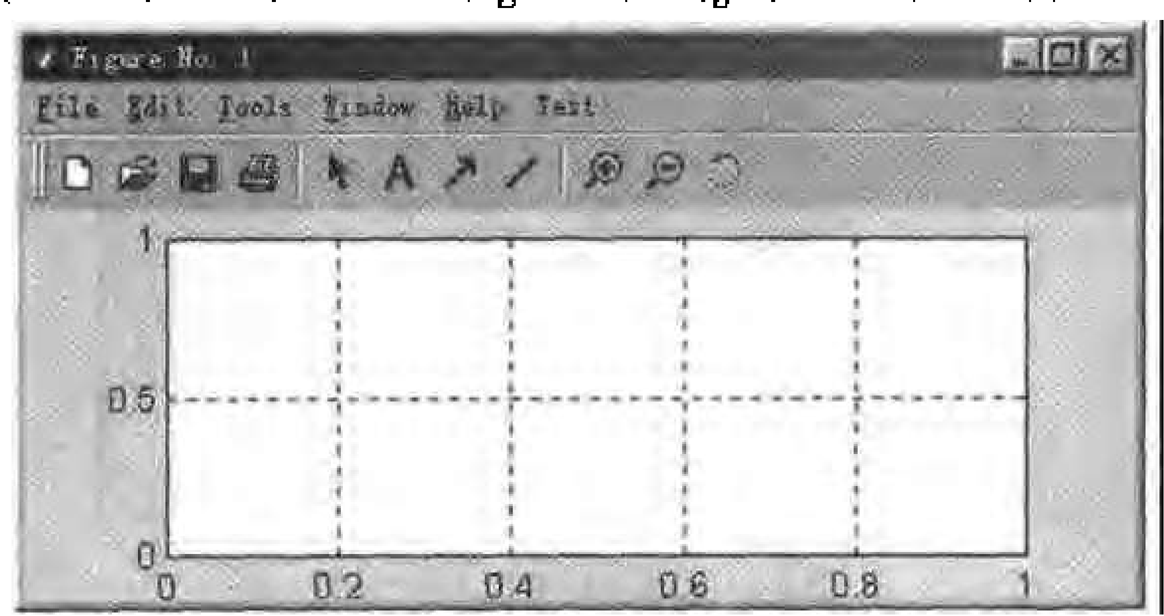


图 3-10　通过顶层菜单 Test 形成的带分格的封闭坐标轴

(4)产生图 3-10 界面的 uimenu 的书写格式之二:方括号续行号表示法

```
uimenu('Label','Test', …
'Callback',['grid on,' , …
  'set(gca,''box'',''on'');'])
```

(5)产生图 3-10 界面的 uimenu 的书写格式之三:串变量法

```
Lpv = 'Test';
Cpv = ['grid on,','set(gca,''box'',''on''),'];
uimenu('Label', Lpv, 'Callback' , Cpv)
```

(6)产生图 3-10 界面的 uimenu 的书写格式之四:构架表示法

```
PS.Label = 'Test';
PS.Callback = ['grid on;','set(gca,''box'',''on'');'];
uimenu(PS)
```

2. 设置简捷键或快捷键

制作出如图3－11所示的菜单，Color菜单项及其下拉的Blue菜单各带一个简捷键，而另一项下拉菜单Red带一个快捷键。

[exm11332 _ 1.m]

```
figure
h_menu = uimenu(gcf,'Label','&Color');
%带简捷键C的用户菜单Color      <2>
h_submenu1 = uimenu(h_menu,'Label','&Blue',…
%带简捷键B的的下拉菜单Blue      <3>
'Callback','set(gcf,''color'',''blue'')');
h_submenu2 = uimenu(h_menu,'label','Red',…
%制作另一个下拉菜单Red
'Callback','set(gcf,''color'',''red'')',…
'Accelerator','r');
%为Red菜单设置快捷键R      <7>
```



图3－11 为用户菜单设置快捷键

3. 用户菜单的外观设计

(A)把用户菜单'Option'设置为顶层的第3菜单项；(B)下拉菜单被两条分隔线分为三个菜单区；(C)最下菜单项又有两个子菜单组成。

(1)编写程序，生成如图3－12所示界面

[exm11333 _ 1.m]

```
figure
h_menu = uimenu('label','Option','Position',3);
h_sub1 = uimenu(h_menu,'label','grid on','callback','grid on');
h_sub2 = uimenu(h_menu,'label','grid off','callback','grid on');
h_sub3 = uimenu(h_menu,'label','box on','callback','box on',…
'separator','on');                                                    %<6>
h_sub4 = uimenu(h_menu,'label','box off','callback','box off');
h_sub5 = uimenu(h_menu,'label','Figure Color','Separator','on');      %<8>
h_subsub1 = uimenu(h_sub5,'label','Red','ForeGroundColor','r',…       %<9>
    'callback','set(gcf,''Color'',''r'')');
```

```
h_subsub2 = uimenu(h_sub5,'label','Reset',…
   'callback','set(gcf,''Color'',''w'')');
```

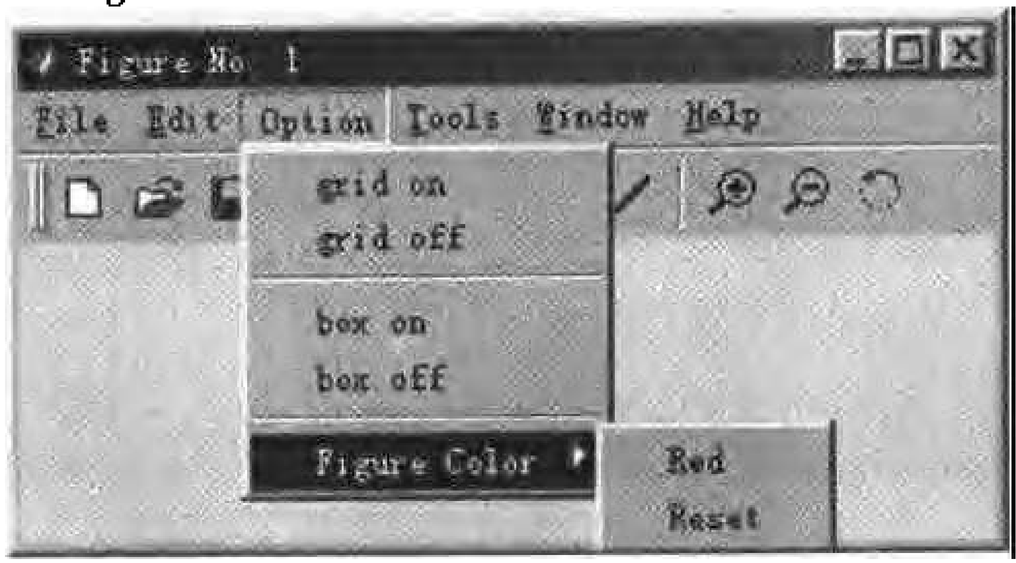


图 3-12　为用户菜单设置外观

(2)位置属性的获取

```
Pos_0 = get(h_menu,'position'),              %查询 Option 菜单位置值
Pos_BoxOn = get(h_sub3,'position')           %查询 box ob 子菜单位置值
Pos_Red = get(h_subsub1,'position')          %查询 red 子菜单的位置值
Pos_0 =
     3
Pos_BoxOn =
     3
Pos_Red =
     1
```

当某菜单项选中后,如何使该菜单项贴上检录符"√"。

[exm11333_2.m]

```
figure
h_menu = uimenu('label','Option');
h_sub1 = uimenu(h_menu,'label','Grid on',…                    % <3>
   'callback',[…
'grid on,',…
'set(h_sub1,''checked'',''on''),',…
'set(h_sub2,''checked'',''off''),',…
]);
h_sub2 = uimenu(h_menu,'label','Grid off',…                   % <4>
'callback',[…
'grid off,',…
'set(h_sub2,''checked'',''on''),',…
'set(h_sub1,''checked'',''off''),',…
]);
```

Grid on 菜单选中后出现检录符如图 3 - 13 所示。

图 3 - 13 Grid on 菜单选中后出现检录符

4. 使能(Enable)与可见性(Visible)属性

制作一个带四个子菜单项的顶层菜单项;该下拉菜单分为两个功能区;每个功能区的两个菜单项是相互对立的,因此采用使能属性处理;当图形窗坐标轴消隐时,整个坐标分隔控制功能区不可见。

(1)编写如下脚本 M 文件 exm11334_1.m

```
[exm11334_1.m]
clf
h_menu = uimenu('label','Option');          %产生顶层菜单项 Option
h_sub1 = uimenu(h_menu,'label','Axis on');  %产生 Axis on 菜单项,由缺省设置而使能
h_sub2 = uimenu(h_menu,'label','Axis off',…
    'enable','off');                         %产生 Axis off 菜单项,但失能
h_sub3 = uimenu(h_menu,'label','Grid on',…
    'separator','on','visible','off');       %产生与上分隔的 Grid on 菜单项,但不可见
h_sub4 = uimenu(h_menu,'label','Grid off',…
    'visible','off');                        %产生 Grid off 菜单项,但不可见
set(h_sub1,'callback',[…                     %选中 Axis on 菜单项后,产生回调操作
    'Axis on,',…                             %画坐标
    'set(h_sub1,''enable'',''off''),',…      %Axis on 菜单项失能
    'set(h_sub2,''enable'',''on''),',…       %Axis off 菜单项使能
    'set(h_sub3,''visible'',''on''),',…      %Grid on 菜单项可见
    'set(h_sub4,''visible'',''on''),']);     %Grid off 菜单项可见
set(h_sub2,'callback',[…                     %%选中 Axis off 菜单项后,产生回调操作
    'axis off,',…                            %使坐标消失
    'set(h_sub1,''enable'',''on''),',…       %Axis on 菜单项使能
    'set(h_sub2,''enable'',''off''),',…      %Axis off 菜单项失能
    'set(h_sub3,''visible'',''off''),',…     %Grid on 菜单项不可见
```

```
    'set(h_sub4,''visible'',''off''),']);      %Grid off 菜单项不可见
set(h_sub3,'callback',[...                    %选中 Grid on 菜单项后,产生回调
'grid on,',...                                %画坐标分格线
'set(h_sub3,''enable'',''off''),',...         %Grid on 菜单项失能
'set(h_sub4,''enable'',''on''),']);           %Grid off 菜单项使能
set(h_sub4,'callback',[...                    %选中 Grid off 菜单项,产生回调
'grid off,',...                               %消除坐标分格线
'set(h_sub3,''enable'',''on''),',...          %Grid on 菜单项使能
'set(h_sub4,''enable'',''off''),']);          %Grid off 菜单项失能
```

(2)在 MATLAB 指令窗中运行 exm11334_1，得到图 3-14 所示的界面

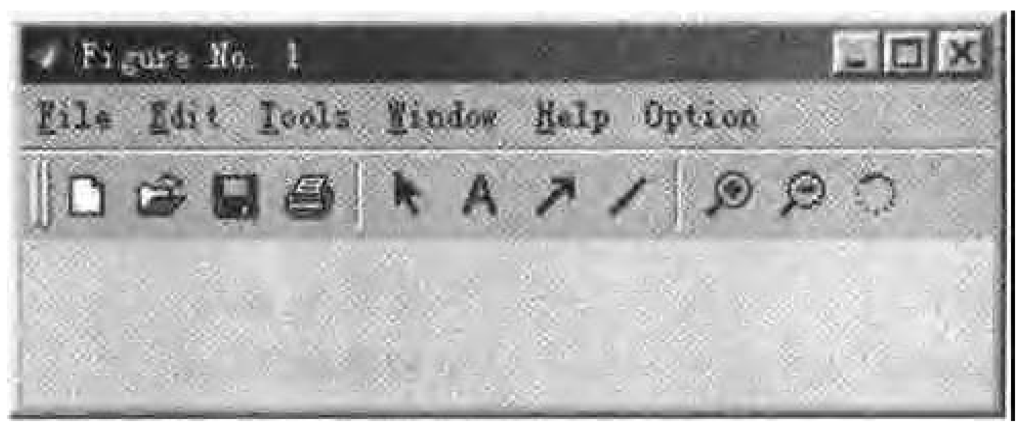


图 3-14　产生顶层菜单项 Option

(3)选中【Option】菜单项,界面呈现如图 3-15 所示



图 3-15　菜单功能的使能性

(4)选中【Option:Axis on】后,界面呈现如图 3-16 所示

(5)选中【Option:Grid on】后,界面呈现如图 3-17 所示



图 3-16　Axis On 菜单项失能



图 3-17 菜单可见性属性

3.2.4 现场菜单的制作

绘制一条 Sa 曲线,创建一个与之相联系的现场菜单,用以控制 Sa 曲线的颜色。

(1)编写脚本 M 文件 exm1134 _ 1.m。

```
[exm1134 _ 1.m]
t = ( - 3 * pi:pi/50:3 * pi) + eps;
y = sin(t)./t;
hline = plot(t,y);                          %绘制 Sa 曲线
cm = uicontextmenu;                         %创建现场菜单
%制作具体菜单项,定义相应的回调
uimenu(cm,'label','Red','callback','set(hline,''color'',''r''),')
uimenu(cm,'label','Blue','callback','set(hline,''color'',''b''),')
uimenu(cm,'label','Green','callback','set(hline,''color'',''g''),')
set(hline,'uicontextmenu',cm) %使 cm 现场菜单与 Sa 曲线相联系
```

(2)在指令窗中运行文件 exm1134 _ 1.m ,得到图 3-18 所示的(但为蓝色的)Sa 曲线。

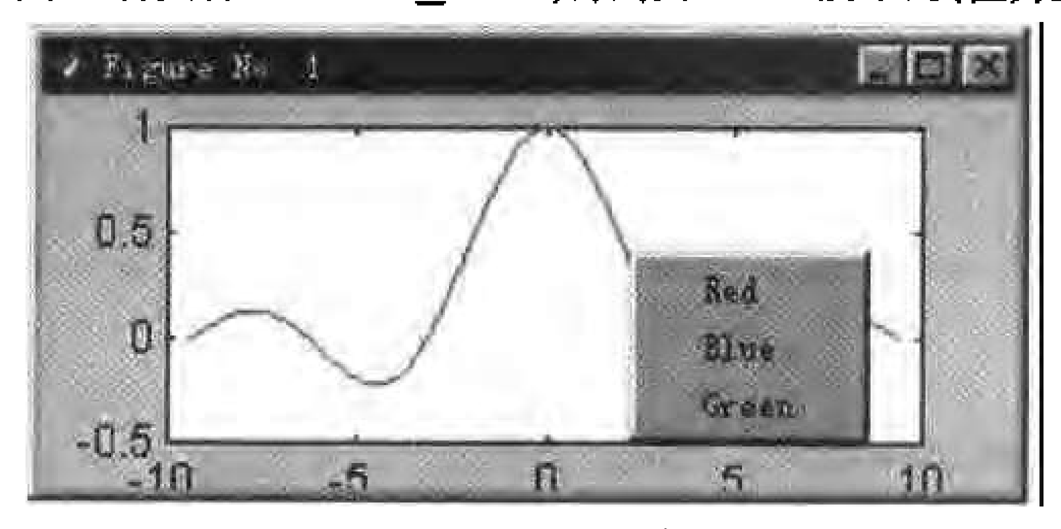


图 3-18 Context 菜单

(3)将鼠标指针指向线条,点击鼠标右键的同时弹出现场菜单,在选中某菜单项(如 Red)后,Sa 曲线就改变(为红)颜色(如图 3-18 所示)。

3.3 用户控件

3.3.1 双位按键、无线电按键、控件区域框示例

创建一个界面包含四种控件:静态文本、"无线电"选择开关、双位按键、控件区域框,

如图3-19所示。

```
[exm11431 _ 1.m]
clf reset
set(gcf,'menubar','none')
set(gcf,'unit','normalized','position',[0.2,0.2,0.64,0.32]);
set(gcf,'defaultuicontrolunits','normalized')      %设置用户缺省控件单位属性值
h _ axes=axes('position',[0.05,0.2,0.6,0.6]);
t=0:pi/50:2*pi;y=sin(t);plot(t,y);
set(h _ axes,'xlim',[0,2*pi]);
set(gcf,'defaultuicontrolhorizontal','left');
htitle=title('正弦曲线');
set(gcf,'defaultuicontrolfontsize',12);       %设置用户缺省控件字体属性值
uicontrol('style','frame',…                   %创建用户控件区              <11>
   'position',[0.67,0.55,0.25,0.25]);
uicontrol('style','text',…                    %创建静态文本框              <13>
   'string','正斜体图名:',…
   'position',[0.68,0.77,0.18,0.1],…
   'horizontal','left');
hr1=uicontrol(gcf,'style','radio',…           %创建"无线电"选择按键        <17>
   'string','正体',…                          %按键功能的文字标识'正体'
   'position',[0.7,0.69,0.15,0.08]);          %按键位置
set(hr1,'value',get(hr1,'Max'));              %因图名缺省使用正体,所以小圆圈应被
                                                点黑                        <20>
set(hr1,'callback',[…                         %                            <21>
set(hr1,''value'',get(hr1,''max'')),',…       %选中将小圆圈点黑            <22>
'set(hr2,''value'',get(hr2,''min'')),',…      %将"互斥"选项点白            <23>
'set(htitle,''fontangle'',''normal''),',…     %使图名字体正体显示
]);
hr2=uicontrol(gcf,'style','radio',…           %创建"无线电"选择按键        <26>
'string','斜体',…                             %按键功能的文字标识'斜体'
'position',[0.7,0.58,0.15,0.08],…             %按键位置
'callback',[…
'set(hr1,''value'',get(hr1,''min'')),',…      %                            <30>
'set(hr2,''value'',get(hr2,''max'')),',…      %                            <31>
'set(htitle,''fontangle'',''italic'')',…      %使图名字体斜体显示
]);                                           %                            <33>
  ht=uicontrol(gcf,'style','toggle',…         %制作双位按键                <34>
'string','Grid',…
'position',[0.67,0.40,0.15,0.12],…
```

```
'callback','grid');
```

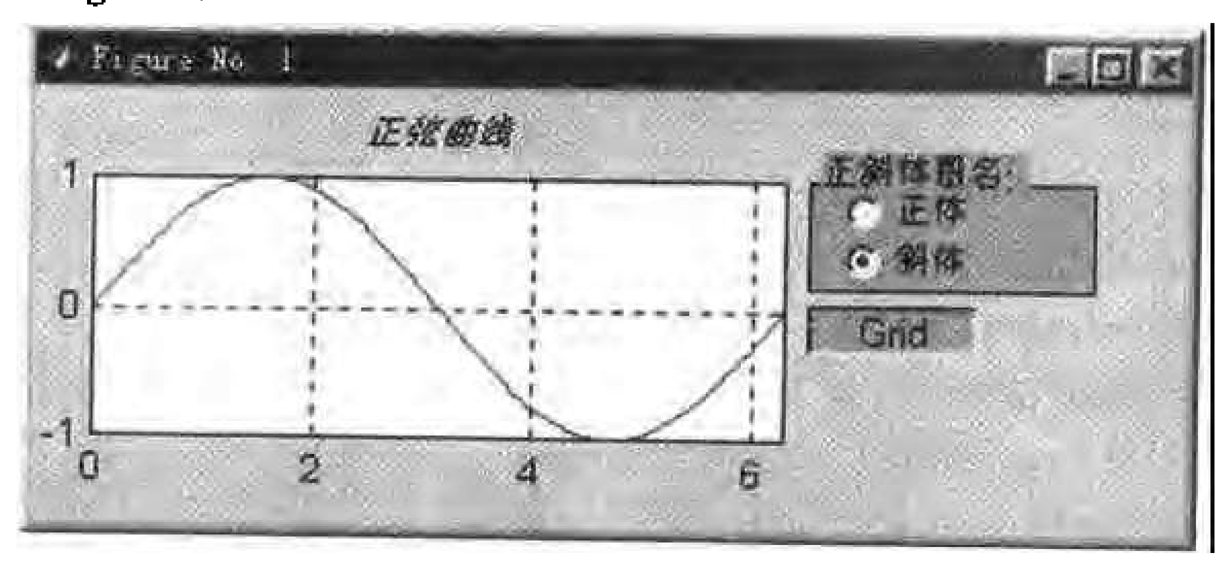


图 3-19 静态文本、选择开关、双位按键及控件区域框

3.3.2 静态文本框、滑动键、检录框示例

制作演示"归一化二阶系统单位阶跃响应"的交互界面。在该界面中，阻尼比可在[0.02,2.02]中连续调节，标志当前阻尼比值；可标志峰值时间和大小；可标志（响应从0到0.95所需的）上升时间。本例涉及以下主要内容：(A)静态文本的创建和实时改写；(B)滑动键的创建，'Max' 和 'Min' 的设置，'Value' 的设置和获取；(C)检录框的创建，'Value' 的获取；(D)受多个控件影响的回调操作。

```
[exm11432_1.m]
clf reset
set(gcf,'unit','normalized','position',[0.1,0.2,0.64,0.35]);
set(gcf,'defaultuicontrolunits','normalized');
set(gcf,'defaultuicontrolfontsize',12);
set(gcf,'defaultuicontrolfontname','隶书');
set(gcf,'defaultuicontrolhorizontal','left');
str='归一化二阶系统阶跃响应曲线';
set(gcf,'name',str,'numbertitle','off');          %书写图形窗名
h_axes=axes('position',[0.05,0.2,0.6,0.7]);      %定义轴位框位置
set(h_axes,'xlim',[0,15]);                        %设置时间轴长度
str1='当前阻尼比=';
t=0:0.1:10;z=0.5;y=step(1,[1 2*z 1],t);
hline=plot(t,y);
htext=uicontrol(gcf,'style','text',...            %制作静态说明文本框   <14>
  'position',[0.67,0.8,0.33,0.1],...
  'string',[str1,sprintf('%1.4g\',z)]);
hslider=uicontrol(gcf,'style','slider',...        %创建滑动键           <17>
'position',[0.67,0.65,0.33,0.1],...
  'max',2.02,'min',0.02,...                       % 设最大阻尼比为2,最小阻尼
                                                    比为0.02            <19>
```

```
    'sliderstep',[0.01,0.05],…                      % 箭头操纵滑动步长 1%,游标
                                                    滑动步长 5%                <20>
    'Value',0.5);                                   %缺省取阻尼比等于 0.5       <21>
hcheck1 = uicontrol(gcf,'style','checkbox',…        %创建峰值检录框            <22>
   'string','最大峰值' ,…
    'position',[0.67,0.50,0.33,0.11]);
vchk1 = get(hcheck1,'value');                       %获得峰值检录框的状态值
                                                                               <25>
hcheck2 = uicontrol(gcf,'style','checkbox',…        %创建上升时间检录框        <26>
   'string','上升时间(0 - >0.95)',…
    'position',[0.67,0.35,0.33,0.11]);
vchk2 = get(hcheck2,'value');                       %获得上升时间检录框的状态值
                                                                               <29>
set(hslider,'callback',[…                           %操作滑动键,引起回调       <30>
    'z = get(gcbo,''value'');',…                    %获得滑动键状态值          <31>
'callcheck(htext,str1,z,vchk1,vchk2)']);            %被回调的函数文件          <32>
set(hcheck1,'callback',[…                           %操作峰值检录框,引起回调
                                                                               <33>
    'vchk1 = get(gcbo,''value'');',…                %获得峰值检录框状态值
                                                                               <34>
    'callcheck(htext,str1,z,vchk1,vchk2)']);        %被回调的函数文件          <35>
set(hcheck2,'callback',[…                           %操作峰值检录框,引起回调
                                                                               <36>
    'vchk2 = get(gcbo,''value'');',…                %获得峰值检录框状态值
                                                                               <37>
    'callcheck(htext,str1,z,vchk1,vchk2)']);        %被回调的函数文件          <38>

[callcheck.m]
function callcheck(htext,str1,z,vchk1,vchk2)
cla,set(htext,'string',[str1,sprintf('%1.4g\ ',z)]);   %更新静态文本框内容
                                                                               <2>
dt = 0.1;t = 0:dt:15;N = length(t);y = step(1,[1 2 * z 1],t);plot(t,y);
if vchk1                                            %假如峰值框被选中          <4>
[ym,km] = max(y);
   if km < (N - 3)                                  %假如在设定时间范围内能插值
                                                                               <6>
k1 = km - 3;k2 = km + 3;k12 = k1:k2;tt = t(k12);
yy = spline(t(k12),y(k12),tt);                      %局部样条插值              <8>
[yym,kkm] = max(yy);                                %求更精确的峰值位置
```

```
line(tt(kkm),yym,'marker','.',...                     %画峰值点            <10>
  'markeredgecolor','r','markersize',20);
ystr=['ymax = ',sprintf('%1.4g \ ',yym)];
tstr=['tmax = ',sprintf('%1.4g \ ',tt(kkm))];
text(tt(kkm),1.05*yym,{ystr;tstr})
   else                                               %假如在设定时间范围内不能插
                                                        值                  <15>
text(10,0.4*y(end),{'ymax --> 1';'tmax --> inf'})
   end
end
if vchk2                                              %假如上升时间框被选中
                                                                            <19>
   k95=min(find(y>0.95));k952=[(k95-1),k95];
   t95=interp1(y(k952),t(k952),0.95);                 %线性插值            <21>
  line(t95,0.95,'marker','o','markeredgecolor','k','markersize',6);
   tstr95=['t95 = ',sprintf('%1.4g \ ',t95)];
   text(t95,0.65,tstr95)
end
```

运行结果如图 3-20 所示。

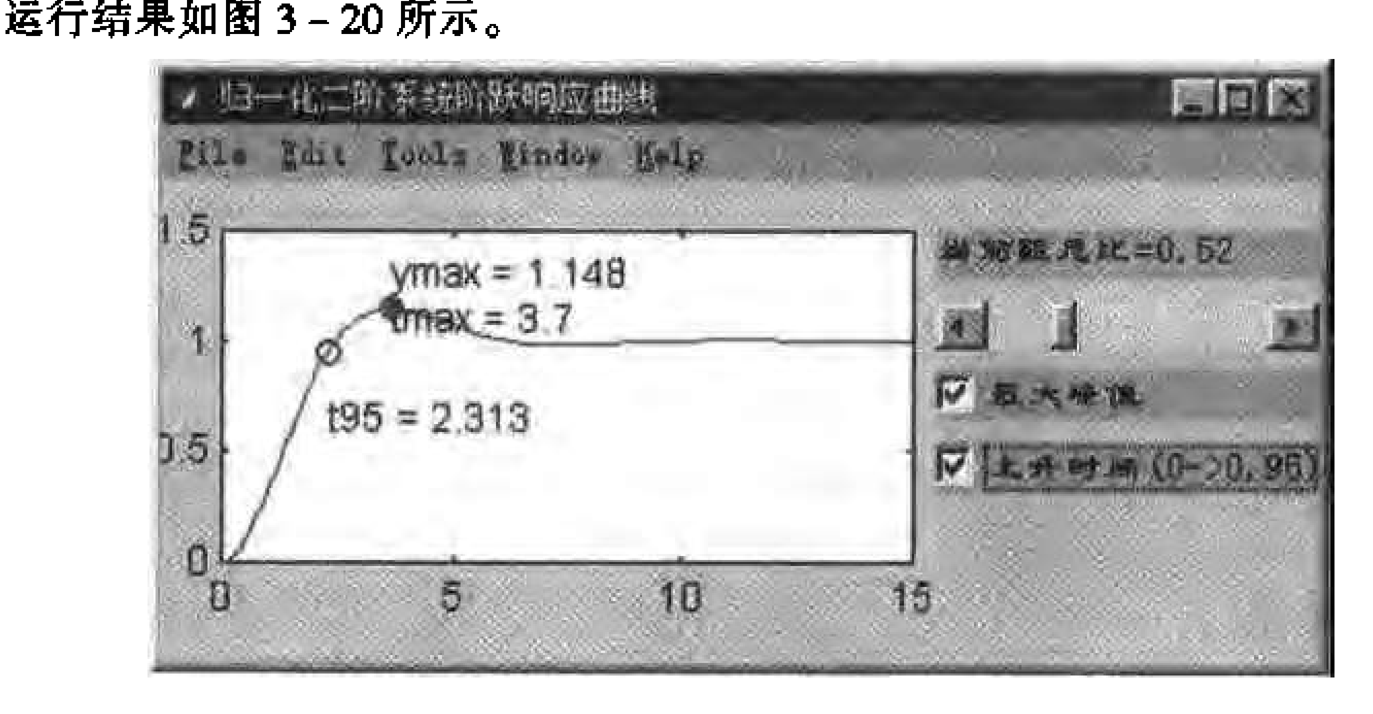


图 3-20 静态文本框、滑动键、检录框

3.3.3 可编辑框、弹出框、列表框、按键示例

制作一个能绘制任意图形的交互界面。它包括:可编辑文本框、弹出框、列表框。本例的关键内容是:如何使编辑框允许输入多行指令。

```
[exm11433_1.m]
clf reset                                                                   <1>
set(gcf,'unit','normalized','position',[0.1,0.4,0.85,0.35]);   %设置图形窗大小
set(gcf,'defaultuicontrolunits','normalized');
```

```
set(gcf,'defaultuicontrolfontsize',11);
set(gcf,'defaultuicontrolfontname','隶书');
set(gcf,'defaultuicontrolhorizontal','left');
set(gcf,'menubar','none');                                   %删除图形窗工具条
str='通过多行指令绘图的交互界面';
set(gcf,'name',str,'numbertitle','off');                     %书写图形窗名
h_axes=axes('position',[0.05,0.15,0.45,
0.70],'visible','off');                                      %定义轴位框位置
uicontrol(gcf,'Style','text',…                               %制作静态文本框
   'position',[0.52,0.87,0.26,0.1],…
   'String','绘图指令输入框');
hedit=uicontrol(gcf,'Style','edit',…                         %制作可编辑文本框      <14>
'position',[0.52,0.05,0.26,0.8],…
   'Max',2);                                                 %取 2,使 Max-Min>1,而允许
                                                               多行输入              <16>
hpop=uicontrol(gcf,'style','popup',…                         %制作弹出菜单          <17>
   'position',[0.8,0.73,0.18,0.12],…
   'string','spring|summer|autumn|winter');                  %设置弹出框中选项名    <19>
hlist=uicontrol(gcf,'Style','list',…                         %制作列表框            <20>
'position',[0.8,0.23,0.18,0.37],…
   'string','Grid on|Box on|Hidden off|Axis off',…           %设置列表框中选项名
                                                                                    <22>
   'Max',2);                                                 %取 2,使 Max-Min>1,而允许
                                                               多项选择              <23>
hpush=uicontrol(gcf,'Style','push',…                         %制作与列表框配用的按键
                                                                                    <24>
   'position',[0.8,0.05,0.18,0.15],'string','Apply');
set(hedit,'callback','calledit(hedit,hpop,hlist)');          %编辑框输入引起回调    <26>
set(hpop,'callback','calledit(hedit,hpop,hlist)');           %弹出框选择引起回调
                                                                                    <27>
set(hpush,'callback','calledit(hedit,hpop,hlist)');          %按键引起的回调        <28>
```

[calledit.m]

```
function calledit(hedit,hpop,hlist)
ct=get(hedit,'string');                                      %获得输入的字符串函数
                                                                                    <2>
vpop=get(hpop,'value');                                      %获得选项的位置标识    <3>
vlist=get(hlist,'value');                                    %获得选项位置向量      <4>
if ~isempty(ct)                                              %可编辑框输入非空时    <5>
```

```
eval(ct')                                              %运行从编辑文本框送人的指令
                                                                                <6>
  popstr = {'spring','summer','autumn','winter'};   %弹出框色图矩阵          <7>
  liststr = {'grid on','box on','hidden off','axis off'}; %列表框选项内容    <8>
  invstr = {'grid off','box off','hidden on','axis on'}; %列表框的逆指令     <9>
  colormap(eval(popstr{vpop}))                      %采用弹出框所选色图      <10>
 vv = zeros(1,4);vv(vlist) = 1;
  for k = 1:4
if vv(k);eval(liststr{k});else eval(invstr{k});end  %按列表选项影响图形
  end
end
```

运行结果如图 3-21 所示。



图 3-21 可编辑框、弹出框、列表框、按键

3.4 由 M 函数文件产生用户菜单和控件

3.4.1 利用全局变量编写用户界面函数文件

利用 M 函数文件创建与图 3-21 相同的用户界面。本例演示:如何依靠全局变量传递控件的图柄,从而保证回调动作正确执行。

(1)编写 M 函数文件 exm1151 _ 1.m 和 calledit1.m。

[exm1151 _ 1.m]

```
function exm1151_1( )
global hedit hpop hlist
(这中间是:原 exm11433 _ 1.m 第<1>行到第<25>行的全部指令)
set(hedit,'callback','calledit1');          %编辑框输入引起回调     <26>
set(hpop,'callback','calledit1');           %弹出框选择引起回调     <27>
set(hpush,'callback','calledit1');          %按键引起的回调         <28>
```

[calledit1.m]

```
function calledit1( )
global hedit hpop hlist
```

(下面续接内容是:原 calledit.m 第<2>行以下的全部指令)

(2)在 MATLAB 指令窗中运行 exm1151_1 就可获得题目所要求的图形用户界面。

3.4.2 利用 'UserData' 属性编写用户界面函数文件

利用 M 函数文件创建与图 3-21 相同的用户界面。本例演示:如何依靠图形窗的'UserData'属性传送用户控件的图柄,从而保证回调动作正确执行。

(1)编写 M 函数文件 exm1152_1.m 和 calledit2.m。

[exm1152_1.m]

```
function exm1152_1( )
```

(这中间是:原 exm11433_1.m 第<1>行到第<25>行的全部指令)

```
set(hedit,'callback','calledit2');   %编辑框输入引起回调              <26>
set(hpop,'callback','calledit2');    %弹出框选择引起回调              <27>
set(hpush,'callback','calledit2');   %按键引起的回调                  <28>
set(gcf,'UserData',[hedit,hpop,hlist])
```

[calledit2.m]

```
function calledit2( )
H = get(gcf,'UserData');
ct = get(H(1),'string');     %获得输入的字符串函数                   <2>
vpop = get(H(2),'value');    %获得选项的位置标识                     <3>
vlist = get(H(3),'value');   %获得选项位置向量                       <4>
```

(下面续接内容是:原 calledit.m 第<5>行以下的全部指令)

(2)在 MATLAB 指令窗中运行 exm1152_1 就可获得题目所要求的图形用户界面。

3.4.3 利用递归法编写用户界面函数文件

1. 利用 M 函数文件创建与图 3-21 相同的用户界面

本例演示:如何依靠图形窗'UserData'属性在递归调用中传送用户控件的图柄,保证回调动作正确执行。

(1)编写 M 函数文件 exm1153_1.m。

[exm1153_1.m]

```
function exm1153_1(flag)
if nargin < 1;flag = 'startup';end%允许在无输入宗量形式下调用该函数    <2>
if ~ ischar(flag);error('flag must be character ''startup''.');end
switch flag                                        %切换控制            <4>
case 'startup'                                     %                    <5>
clf reset                                          %                    <6>
set(gcf,'unit','normalized','position',[0.1,0.4,0.85,0.35]);
```

```
set(gcf,'defaultuicontrolunits','normalized');
set(gcf,'defaultuicontrolfontsize',11);
set(gcf,'defaultuicontrolfontname','隶书');
set(gcf,'defaultuicontrolhorizontal','left');
set(gcf,'menubar','none');                          %删除图形窗工具条
str='通过多行指令绘图的交互界面';
set(gcf,'name',str,'numbertitle','off');            %书写图形窗名
h_axes=axes('position',[0.05,0.15,0.45,0.70],'visible','off');
uicontrol(gcf,'Style','text',...                    %制作静态文本框
   'position',[0.52,0.87,0.26,0.1],...
   'String','绘图指令输入框');
hedit=uicontrol(gcf,'Style','edit',...              %制作可编辑文本框     <19>
  'position',[0.52,0.05,0.26,0.8],...               %                   <20>
   'Max',2);                                        %取2,使Max-Min>1,而允许
                                                      多行输入          <21>
hpop=uicontrol(gcf,'style','popup',...              %制作弹出菜单         <22>
  'position',[0.8,0.73,0.18,0.12],...               %                   <23>
  'string','spring|summer|autumn|winter');          %设置弹出框中选项名
                                                                        <24>
hlist=uicontrol(gcf,'Style','list',...              %制作列表框           <25>
'position',[0.8,0.23,0.18,0.37],...                 %                   <26>
  'string','Grid on|Box on|Hidden off|Axis off',...%设置列表框中选项名   <27>
  'Max',2);                                         %取2,使Max-Min>1,而允许
                                                      多项选择          <28>
hpush=uicontrol(gcf,'Style','push',...              %制作与列表框配用的按键
                                                                        <29>
  'position',[0.8,0.05,0.18,0.15],'string','Apply');
set(hedit,'callback','exm1153_1(''set'')');         %编辑框输入引起回调   <31>
set(hpop,'callback','exm1153_1(''set'')');          %弹出框选择引起回调   <32>
set(hpush,'callback','exm1153_1(''set'')');         %按键引起的回调       <33>
     set(gcf,'UserData',[hedit,hpop,hlist]);        %向'UserData'存放图柄 <34>
case 'set'                                          %以下是回调函数       <35>
H=get(gcf,'UserData');                              %从'UserData'获取图柄 <36>
ct=get(H(1),'string');                              %获得输入的字符串函数 <37>
vpop=get(H(2),'value');                             %获得选项的位置标识   <38>
vlist=get(H(3),'value');                            %获得选项位置向量     <39>
if ~isempty(ct)
  eval(ct')                                         %运行从编辑文本框送入的指令
  popstr={'spring','summer','autumn','winter'};          %弹出框色图矩阵
```

```
    liststr = {'grid on','box on','hidden off','axis off'};       %列表框选项内容
    invstr = {'grid off','box off','hidden on','axis on'};        %列表框的逆指令
    colormap(eval(popstr{vpop}))                                  %采用弹出框所选色图
    vv = zeros(1,4);vv(vlist) = 1;
    for k = 1:4
if vv(k);eval(liststr{k});else eval(invstr{k});end   %按列表选项影响图形
    end
end                                                                        <50>
end
```

(2)在 MATLAB 指令窗中运行 exm1153 _ 1 就可获得题目所要求的图形用户界面(即图 3-21 无图形时的初始界面)。

2. 利用 M 函数文件创建与图 3-21 相同的用户界面

本例演示:如何依靠 'Tag'属性 与 findobj 指令的配合使用获取回调操作所必需的控件图柄,保证回调动作正确执行。

本例的程序可由 exm1153 _ 1.m 做如下修改而得。

(1)删去 exm1153 _ 1.m 的指令 <34> <36>。

(2)在 exm1153 _ 1.m 的 <20> 和 <21> 行之间增添一行

```
'Tag','H _ edit',…
```

(3)在 exm1153 _ 1.m 的 <23> 和 <24> 行之间增添一行

```
'Tag','H _ popup',…
```

(4)在 exm1153 _ 1.m 的 <27> 和 <28> 行之间增添一行

```
'Tag','H _ list',…
```

(5)把 exm1153 _ 1.m 的 <31> <32> <33> 条指令中的 exm1153 _ 1 改为 exm1153 _ 2。

(6)在 exm1153 _ 1.m 的 <35> 和 <37> 行之间增添以下三条指令。

```
H(1) = findobj(gcf,'Tag','H _ edit');
H(2) = findobj(gcf,'Tag','H _ popup');
H(3) = findobj(gcf,'Tag','H _ list');
```

(7)把 exm1153 _ 1.m 的函数头修改为

```
function exm1153 _ 2(flag)
```

(8)把修改后的文件"另存为"exm1153 _ 2.m,就完成了新文件的编写。

(9)在 MATLAB 指令窗中运行 exm1153 _ 2 就可获得题目所要求的图形用户界面。

3.5 图形用户界面设计工具

图 3-22 是图形化的向导工具面板。

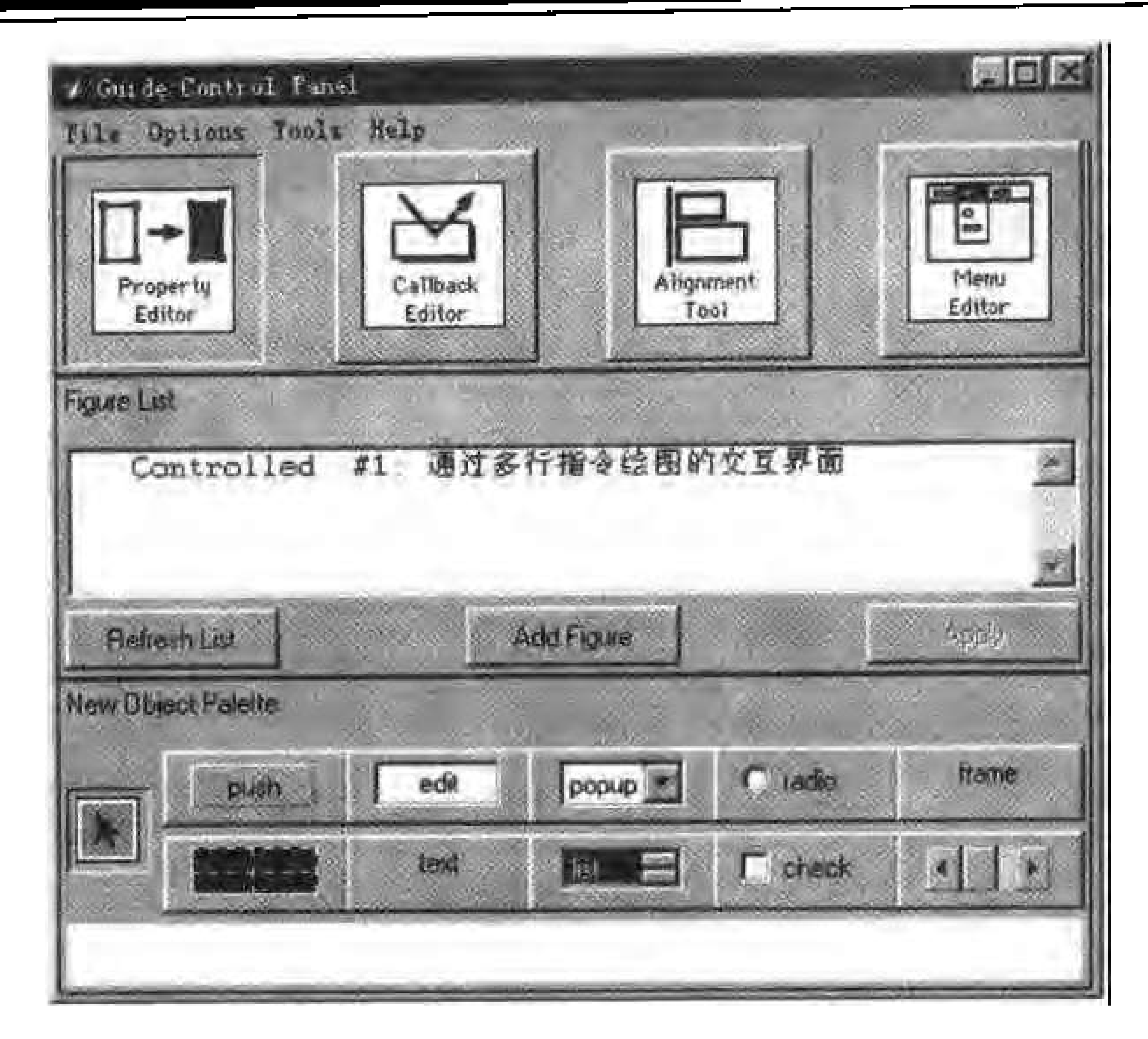


图 3－22 图形化的工具界面

3.5.1 交互式用户界面设计工具应用示例

待制作的用户界面如图 3－23 所示。

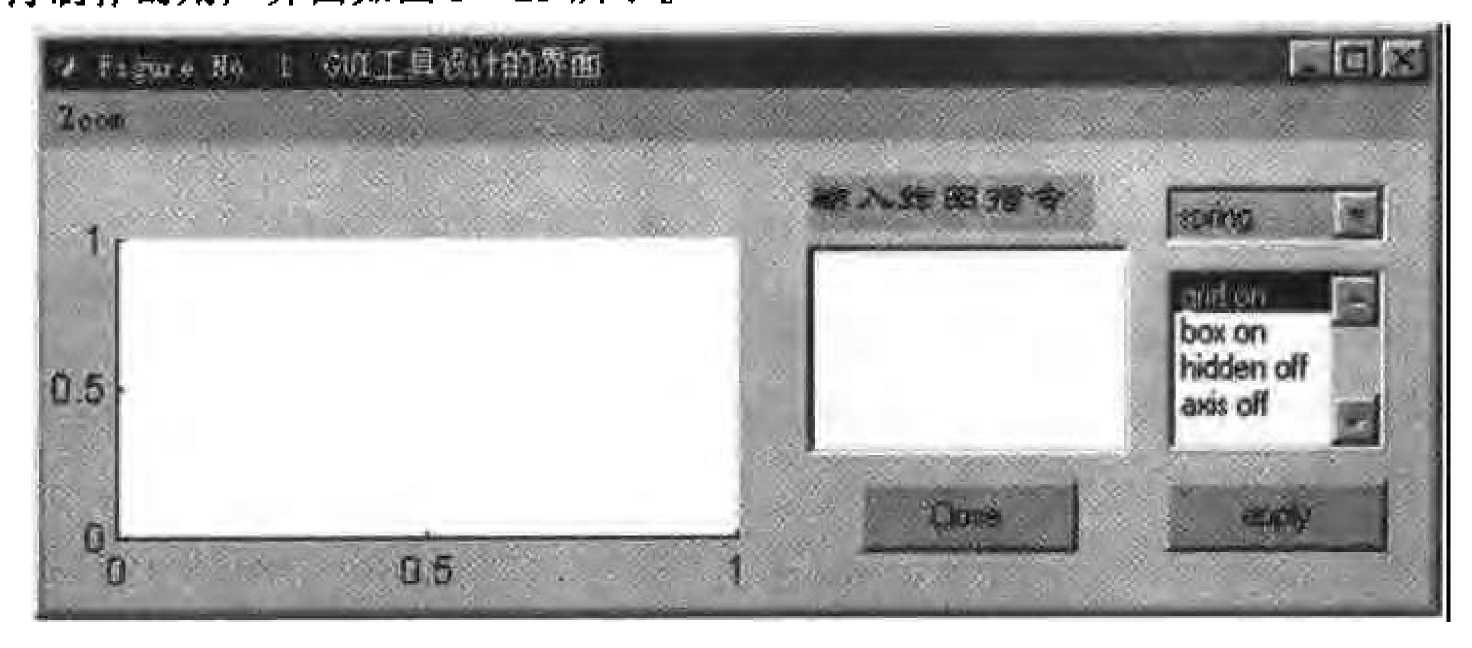


图 3－23 待制作的用户界面

1．工序一：窗口初始位置和大小设计

界面设计工具 guide 的启动和用户界面窗口初试几何制作。属性编辑工具界面如图 3－24 所示。

2．工序二：对象的几何布局

整个用户界面的几何布局如图 3－25 所示。"轴"、控件种类、相对位置及大致尺寸。本例演示：(A)设计工具控制面板上"新对象模块区"的图标的使用。(B)几何布局时不必

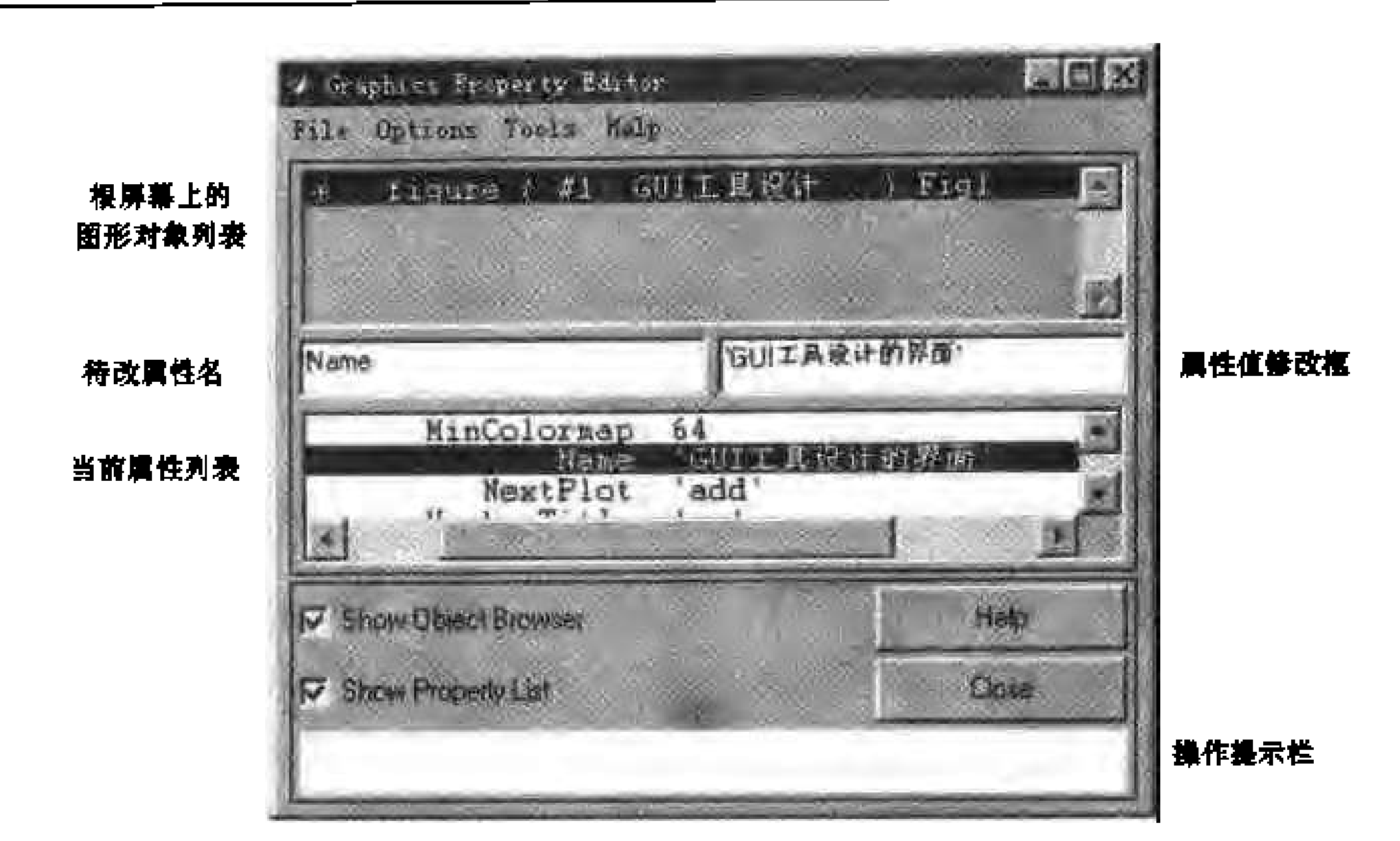


图3-24 属性编辑工具界面

太多考虑各对象的精细位置和大小。如图3-25所示。

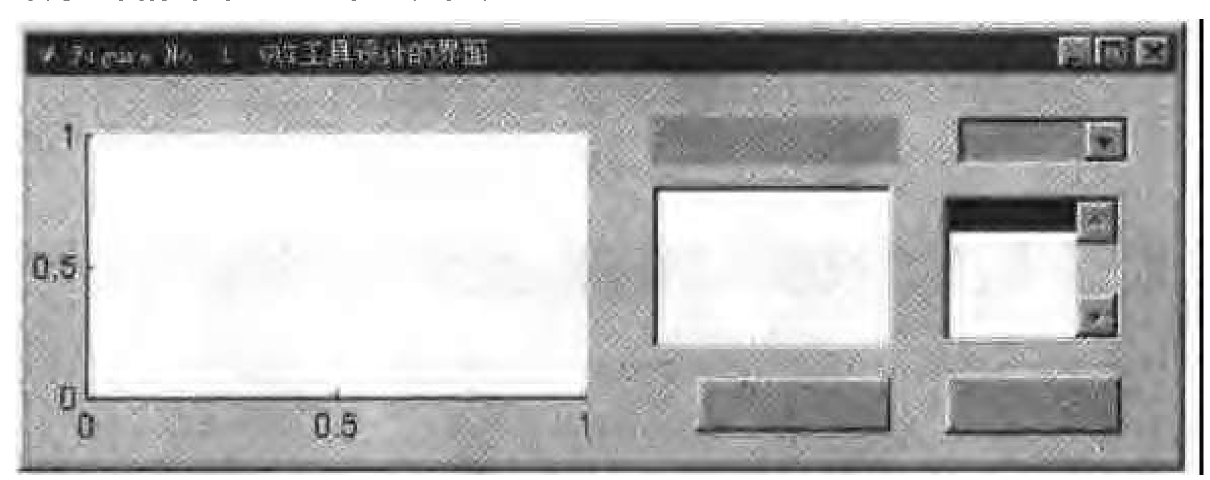


图3-25 “第二道工序”构成控件布局的界面

3. 工序三:新建对象的属性设置

控件关键属性的设置。本例演示:(A)属性编辑工具的使用。(B)当 'Callback' 属性值可用比较简单的 MATLAB 语句表达时,则直接填写;如果语句较多,表达复杂,那么就应采用一个待写的 M 函数名填写。本例中的回调都借助 M 函数文件实现。(C)当控件上有字符串标识时,应注意文字的对齐方式和注意字体大小,使外观上与对象大小协调。(D)控件的 'String' 属性字符串的输入格式。在这过程中,可能还要适当调整对象几何尺寸,使字符表现清晰醒目。(E) 'Units' 采用 'normalized' ,使得所有新建对象随所在图形窗按比例缩放。

4. 工序四:用户菜单的制作

用户菜单制作工序比较独立,因此该工序可前可后,也可以与“工序一”相合并。在本例中菜单引起的回调都是直接、简单的 MATLAB 语句。用户菜单编辑器界面如图3-26所示。

5. 工序五:新建图形对象的齐整化

演示“对齐编辑工具”的使用,如图3-27所示。

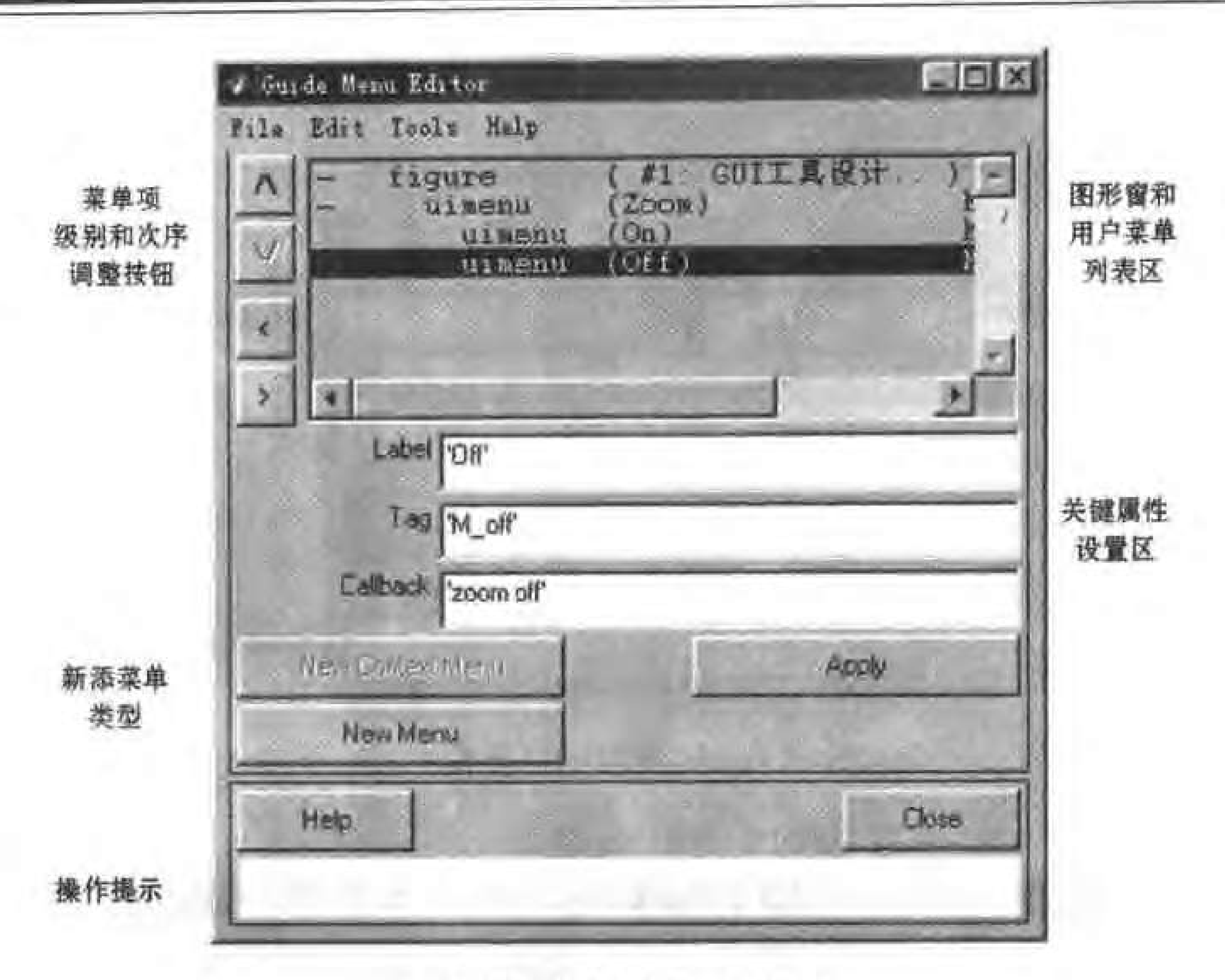


图 3－26 用户菜单编辑器界面

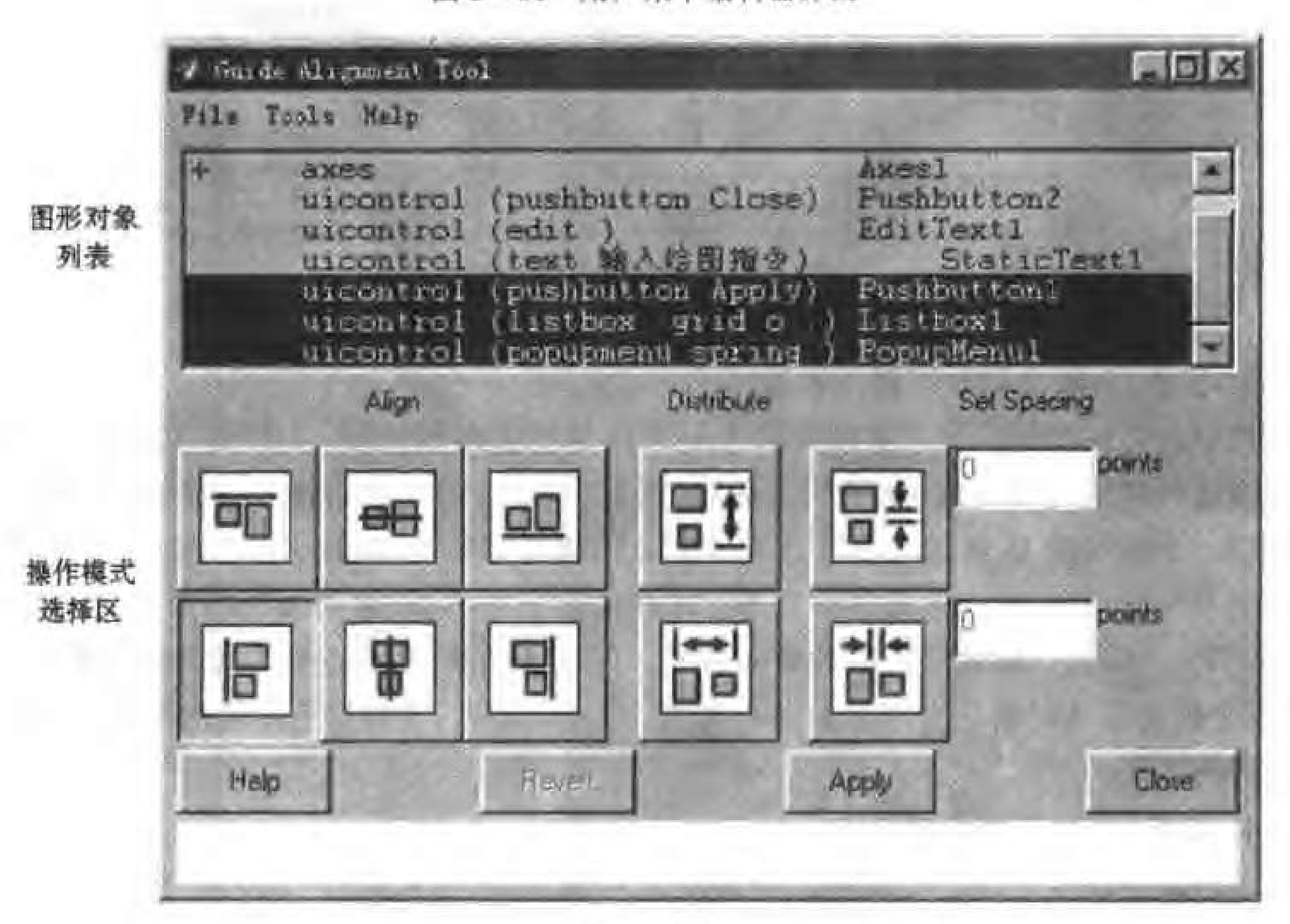


图 3－27

6. 工序六：回调函数的编写

从处理方便出发编写回调函数（关于回调函数的详细讨论，请看 MATLAB 的帮助文档）。

(1)弹出框的回调函数 Mycolormap.m。

```
[Mycolormap.m]
function Mycolormap
popstr={'spring','summer','autumn','winter'};              %弹出框色图矩阵
vpop=get(findobj(gcf,'Tag','PopupMenu1'),'value');  %获得选项的位置标识
colormap(eval(popstr{vpop}))                               %采用弹出框所选色图
```

(2)列表框和“Apply”按键配合的回调函数 Myapply.m。

```
[Myapply.m]
function Myapply
vlist=get(findobj(gcf,'Tag','Listbox1'),'value');          %获得选项位置向量
liststr={'grid on','box on','hidden off','axis off'};      %列表框选项内容
invstr={'grid off','box off','hidden on','axis on'};       %列表框的逆指令
vv=zeros(1,4);vv(vlist)=1;
for k=1:4
   if vv(k);eval(liststr{k});else eval(invstr{k});end      %按列表选项影响图形
end
```

(3)动态编辑框的回调函数 Myedit.m。

```
[Myedit.m]
function Myedit
ct=get(findobj(gcf,'Tag','EditText1'),'string');
eval(ct')
```

7. 工序七:界面功能的全面测试

这个读者可以自己在程序中测试,这里就不再讨论了。

3.5.2 为读者提供的配套文件和数据

(1)机器自动生成的主控文件

```
[Mygui1.m]
function fig = Mygui1()
% This is the machine-generated representation of a Handle Graphics object
% and its children. Note that handle values may change when these objects
% are re-created. This may cause problems with any callbacks written to
% depend on the value of the handle at the time the object was saved.
% This problem is solved by saving the output as a FIG-file.%
% To reopen this object, just type the name of the M-file at the MATLAB
% prompt. The M-file and its associated MAT-file must be on your path.%
% NOTE: certain newer features in MATLAB may not have been saved in this
% M-file due to limitations of this format, which has been superseded by
% FIG-files. Figures which have been annotated using the plot editor tools
% are incompatible with the M-file/MAT-file format, and should be saved as
```

```
% FIG - files.
load Mygui1
h0 = figure('Units','normalized', ...
'Color',[0.8553876799929505 0.8553876799929505 0.8553876799929505], ...
'Colormap',mat0, ...
'FileName','F:\99\m5\Mygui1.m', ...
'MenuBar','none', ...
'Name','GUI 工具设计的界面', ...
'PaperPosition',[18 180 575.9999999999999 432], ...
'PaperUnits','points', ...
'Position',[0.1484375 0.5291666666666667 0.8 0.35], ...
'Tag','Fig1', ...
'ToolBar','none');
h1 = uimenu('Parent',h0, ...
'Label','Zoom', ...
'Tag','M_Z');
h2 = uimenu('Parent',h1, ...
'Callback','zoom on', ...
'Label','On', ...
'Tag','M_Zon');
h2 = uimenu('Parent',h1, ...
'Callback','zoom off', ...
'Label','Off', ...
'Tag','M_Zoff');
h1 = uicontrol('Parent',h0, ...
'Units','normalized', ...
'BackgroundColor',[0.7529411764706 0.7529411764706 0.7529411764706], ...
'Callback','Mycolormap', ...
'ListboxTop',0, ...
'Position',[0.80859375 0.773809523809524 0.16 0.130952380952381], ...
'String',['spring ';'summer ';'autumn ';'winter '], ...
'Style','popupmenu', ...
'Tag','PopupMenu1', ...
'Value',1);
h1 = uicontrol('Parent',h0, ...
'Units','normalized', ...
'BackgroundColor',[1 1 1], ...
'Max',2, ...
'Position',[0.80859375 0.327380952380952 0.16 0.398809523809524], ...
```

```
'String',['grid on    ';'box on     ';'hidden off';'axis off   '], …
'Style','listbox', …
'Tag','Listbox1', …
'Value',1);
h1 = uicontrol('Parent',h0, …
'Units','normalized', …
'BackgroundColor',[.752941176470588 .752941176470588 .752941176470588], …
'Callback','Myapply', …
'ListboxTop',0, …
'Position',[0.80859375 0.12 0.16 0.15], …
'String','apply', …
'Tag','Pushbutton1');
h1 = uicontrol('Parent',h0, …
'Units','normalized', …
'BackgroundColor',[1 1 1], …
'Callback','Myedit', …
'FontName','Times New Roman', …
'FontSize',10, …
'HorizontalAlignment','left', …
'ListboxTop',0, …
'Max',2, …
'Position',[.55078125 .3273809523809524 .232421875 .4523809523809523], …
'Style','edit', …
'Tag','EditText1');
h1 = axes('Parent',h0, …
'CameraUpVector',[0 1 0], …
'CameraUpVectorMode','manual', …
'Color',[1 1 1], …
'ColorOrder',mat1, …
'Position',[0.05 0.15 0.45 0.65], …
'Tag','Axes1', …
'XColor',[0 0 0], …
'YColor',[0 0 0], …
'ZColor',[0 0 0]);
h2 = text('Parent',h1, …
'Color',[0 0 0], …
'HandleVisibility','off', …
'HorizontalAlignment','center', …
'Position',[0.4978165938864628 -0.2222222222222221 9.160254037844386], …
```

```
'Tag','Axes1Text4', …
'VerticalAlignment','cap');
set(get(h2,'Parent'),'XLabel',h2);
h2 = text('Parent',h1, …
'Color',[0 0 0], …
'HandleVisibility','off', …
'HorizontalAlignment','center', …
'Position',[-0.1353711790393013 0.4907407407407408 9.160254037844386], …
'Rotation',90, …
'Tag','Axes1Text3', …
'VerticalAlignment','baseline');
set(get(h2,'Parent'),'YLabel',h2);
h2 = text('Parent',h1, …
'Color',[0 0 0], …
'HandleVisibility','off', …
'HorizontalAlignment','right', …
'Position',mat2, …
'Tag','Axes1Text2', …
'Visible','off');
set(get(h2,'Parent'),'ZLabel',h2);
h2 = text('Parent',h1, …
'Color',[0 0 0], …
'HandleVisibility','off', …
'HorizontalAlignment','center', …
'Position',[0.4978165938864628 1.064814814814815 9.160254037844386], …
'Tag','Axes1Text1', …
'VerticalAlignment','bottom');
set(get(h2,'Parent'),'Title',h2);
h1 = uicontrol('Parent',h0, …
'Units','normalized', …
'FontName','隶书', …
'FontSize',13, …
'HorizontalAlignment','left', …
'ListboxTop',0, …
'Position',[0.55 0.8 0.2 0.12], …
'String','输入绘图指令', …
'Style','text', …
'Tag','StaticText1');
h1 = uicontrol('Parent',h0, …
```

```
'Units','normalized', …
'BackgroundColor',[.752941176470588 .752941176470588 .752941176470588], …
'Callback','close(gcbf)', …
'ListboxTop',0, …
'Position',[0.5869921874999999 0.12 0.16 0.15], …
'String','Close', …
'Tag','Pushbutton2');
if nargout > 0, fig = h0; end
```

(2)配套数据文件

[Myguizzy.m]

```
function Myguizzy
%假如 Mygui1.m 所在目录不是 d:\matbook5\mdisk ,那么第<10>条就应做相应的改变。
%一定要保证本函数生成的 Mygui1.mat 与 Mygui1.m 在同一目录。
mat0 = jet(64);
mat1 = [    0       0       1.0000
    0       0.5000      0
    1.0000      0       0
    0       0.7500      0.7500
    0.7500      0       0.7500
    0.7500      0.7500      0
    0.2500      0.2500      0.2500];
mat2 = [-0.1179     1.3056      9.1603];
save d:\matbook5\mdisk\Mygui1                           %<10>
```

(3)如何利用本节所提供的文件产生图3-23所示的界面

- 把本节提供的 Mygui1.m , Myguizzy.m , Mycolormap.m , Myapply.m , Myedit.m 五个文件放在 MATLAB 的搜索路径上。
- 先运行 Myguizzy.m ,创建数据文件 Mygui1.mat 。
- 运行 Mygui1.m ,就可得到符合要求的界面。

第 4 章　Simulink 入门

4.1　Simulink 概述

4.1.1　什么是 Simulink

Simulink 是 MATLAB 提供的实现动态系统建模和仿真的一个软件包。它让用户把精力从编程转向模型的构造。随着学习的不断深入,读者会认识到 Simulink 一个很大的优点是为用户省去了许多重复的代码编写工作,用户就不必一步一步地从最底层开始编写。

启动 Simulink 的方法有很多种,按照 MATLAB 的传统方式,只要在 MATLAB 命令窗口中键入

≫Simulink

这样,一个称为 Simulink Library Browser 的窗口就会出现在桌面上,它的样子如图 4-1所示。

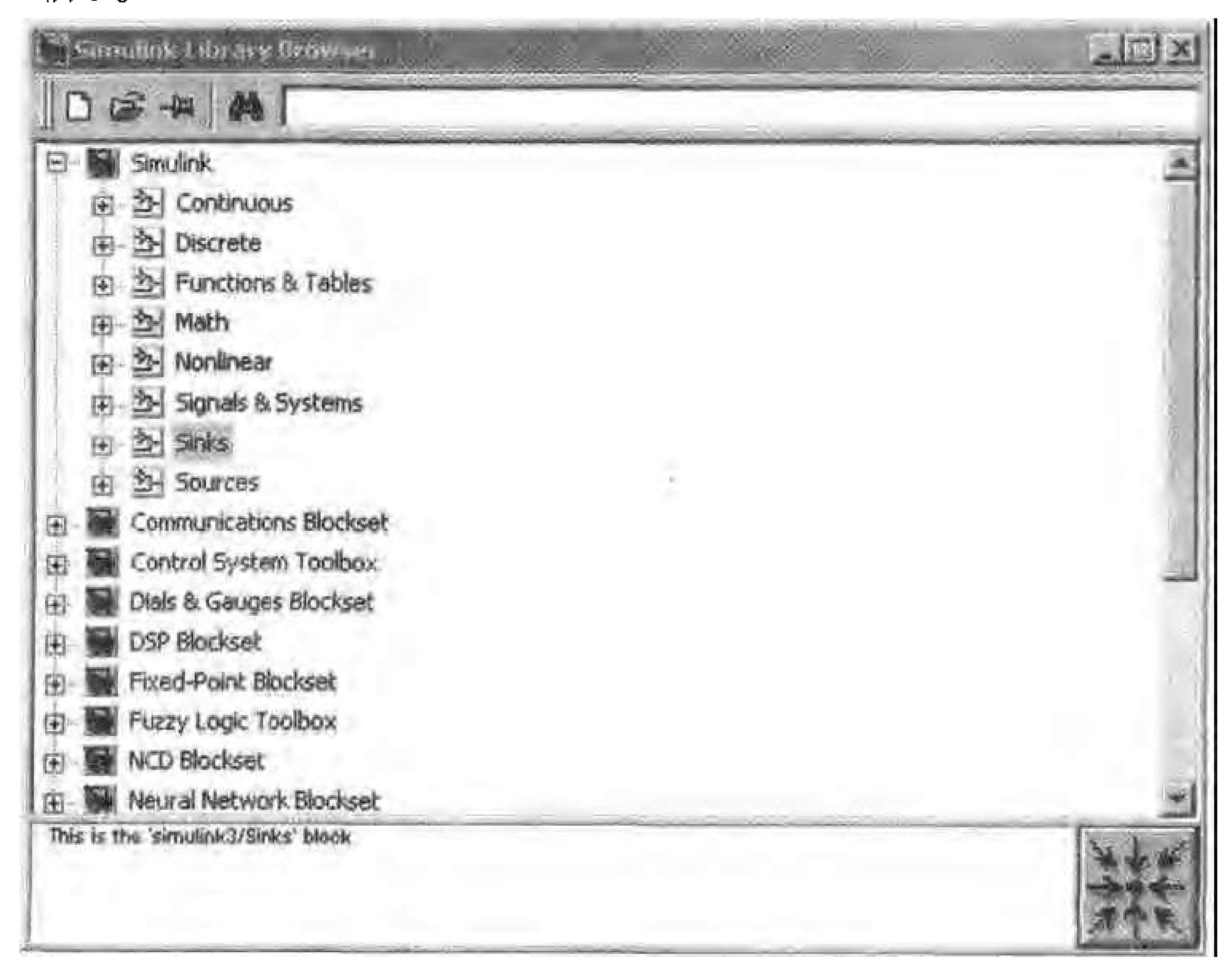


图 4-1　Simulink 库浏览器

读者也可以使用 MATLAB 主窗口的快速按钮或者是 launch pad 子窗口里的命令来打开。本书中使用的版本是 Simulink 3.0(包含在 MATLAB5.3 里)。更高的版本也已经出现了,不过变化不大。

Simulink 让人很振奋的一点就是支持图形用户界面,这比 MATLAB 传统的命令行方式要亲切得多。正是这样,读者就可以自己探索,不一定要按部就班地按照本书的介绍来学习。当然先对 Simulink 工作原理有个整体上的把握还是很有必要的。

前面讲过,Simulink 是用于动态系统建模和仿真的一个软件包,如果把这个过程比作建造房子,那么以前用高级语言或 MATLAB 语言编写的仿真程序的方式就如同是从一堆沙子开始造房子,这不但麻烦,而且有许多重复操作,建造者的精力会大量地浪费在一些相同地例如把沙子变成砖块的事情上,以及如何把它们组在一起变成房子这些技术性的事情,而不能把更多的精力集中到房子的设计上。这在计算机仿真里,就等于是把精力过多地投入到某一个具体的算法的设计上,而不是投入到模型的设计构造本身。Simulink 的目的就是让用户能把更多的精力投入到模型设计本身。它首先提供了一些基本的模块,这些模块就放在上面的库浏览器里,用户可以调用这些模块,而不必再从最基本的做起。Simulink 的每个模块对用户而言都是透明的,用户只需知道模块的输入输出以及模块的功能,而不必管模块内部是怎么实现的。于是,留给用户的事情就是如何连接这些模块来完成自己的仿真任务。连接的方式在 Simulink 里是很简单的,例如要连接两个模块,只需要将一个模块的输入和另一个模块的输出用一根直线连起来就行了。模型构造好之后,用户可以进行仿真,等待结果,或者改变参数,再运行。至于像各个模块在运行时如何执行,时间是如何采样(离散系统),事件是如何驱动等等细节性问题,用户可以根本不用去关心,Simulink 都替你做好了。总之,Simulink 把那些最没有意思、最烦人的细节都屏蔽掉了,而留给用户的是一个友好的环境,让用户以最轻松、最有效的方式完成他们感兴趣的东西。

正如上面所说的,库是存放 Simulink 模块的场所,图 4-1 所示的浏览器窗口就像是一个橱窗,用户可以在里面浏览并选择自己所需要的模块,并可以调用(以后将会看到这只是一个拖动的操作)。下面对图 4-1 做一个简要的说明。

读者的窗口也许和图 4-1 显示的不太一样,这种现象很正常。Simulink 这一标题是必定存在的,它是 Simulink 的基本模块库。而至于通行模块集(Communication blockset)、数字信号处理(DSP blockset)等等,视读者的安装情况而定了,你只有安装了这些工具箱,它们才会出现。

如图 4-1 中所示,主窗口列出了库中的所有模块,它们按类分成各个子库。读者可以点击每个标题前的加号,看看库里面有些什么东西,就如在图 4-1 看到的一样,Simulink 下面分为:Continuous(连续)、Discrete(离散)、Functions & Tables(函数和平台)、Math(数学)、Nonlinear(非线性)、Signals & Systems(信号和系统)、Sinks(接收器)、Sources(源)等子库。可见 Simulink 库是按功能分类的,用户在调用时会很容易就找到自己所需要的模块。读者可以选择这些子库的一个,继续看看它里面的情况。图 4-2 显示的是 Math 子库里的模块。

从模块的名字就可以看出,Math 子库里的模块基本上是一些信号接收器,如 Scope(示波器)、Display(显示器)、XY Graph(XY 图形)等。但如果要满足你的好奇心,看看这

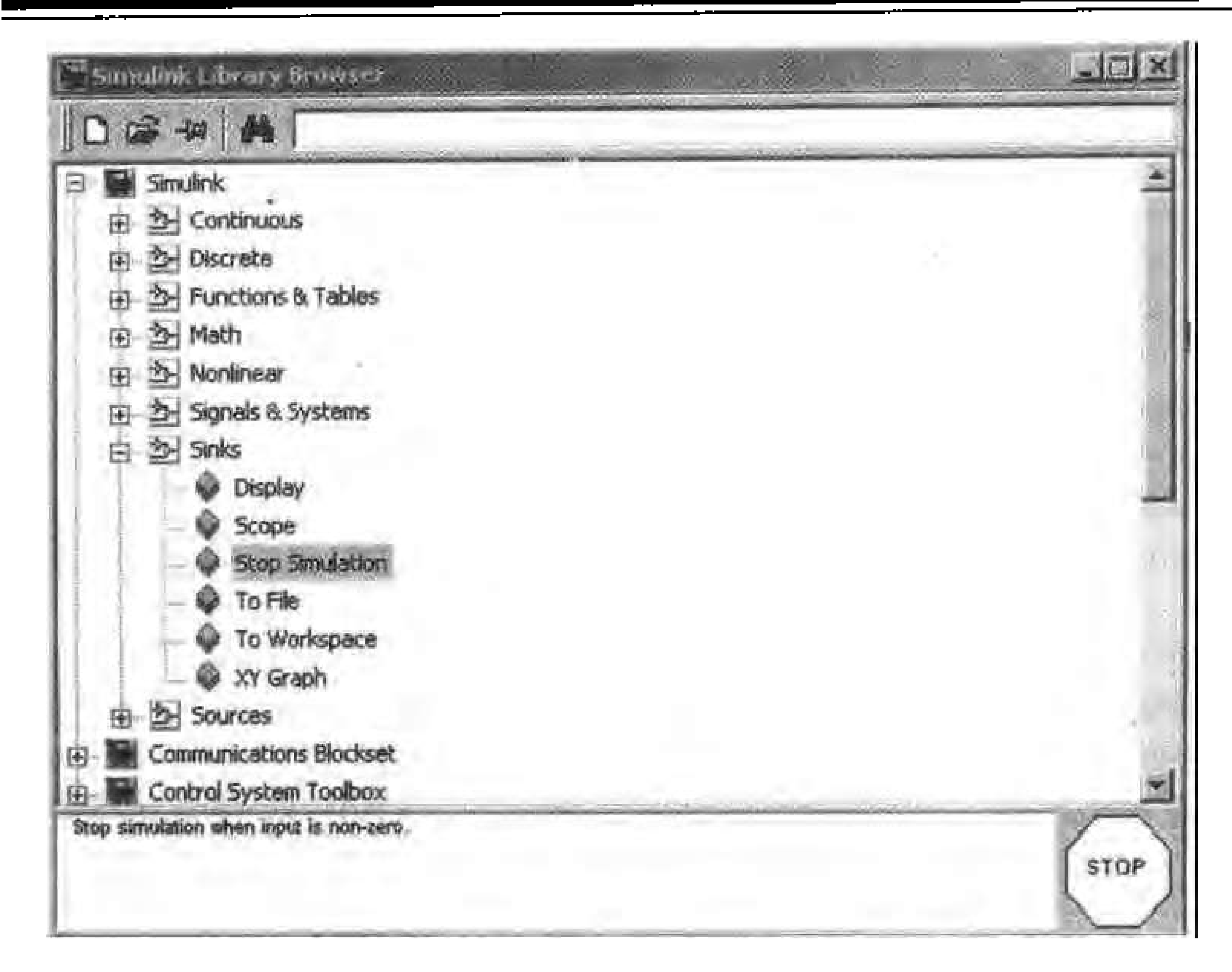


图 4-2 Sinks 子库展开图

些模块，双击这些模块不会起任何作用。

正确的操作是点击工具栏上最左边创建新模块的空白按钮，会出现一个如图 4-3 所示的窗口，不妨称为模型窗口。关于这个窗口的情况在下一节会详细说明。接下来的操作是回到浏览器窗口，在所感兴趣的模块上，比如 constant 模块，按下鼠标左键，然后拖动鼠标至新的窗口上再松开，这时新窗口会出现一个名为 constant 的方块。简言之就是把模块复制到新窗口。图 4-3 显示了这时的情景，图中出现的对话框则是双击该模块后的结果。这个对话框的作用是让用户设计模块的参数，每一个 Simulink 模块都具有这样的参数设置对话框。

创建模型的操作十分简单，读者完全可以独自摸索出来。要点是把你想要的模块拖到用户界面，然后按照你的模型，将它们的输入和输出端口用直线连接起来。

4.1.2 Simulink 模型的特点

下面，让读者体会一下用 Simulink 创建出的模型的样子，并分析它的一些特点。为此在 MATLAB 的命令窗口中输入

```
>>thermo
```

将会出现如图 4-4 所示的窗口，这个窗口和图 4-3 的区别在于图 4-3 的窗口没有包含任何模型，面图 4-4 包含了一个名为 thermo 的模型。这是 Simulink 的一个“房间热力学仿真演示”程序。

读者可以点击执行按钮(如图 4-4 中所示)，或者是 Simulink 菜单下的 start 命令来

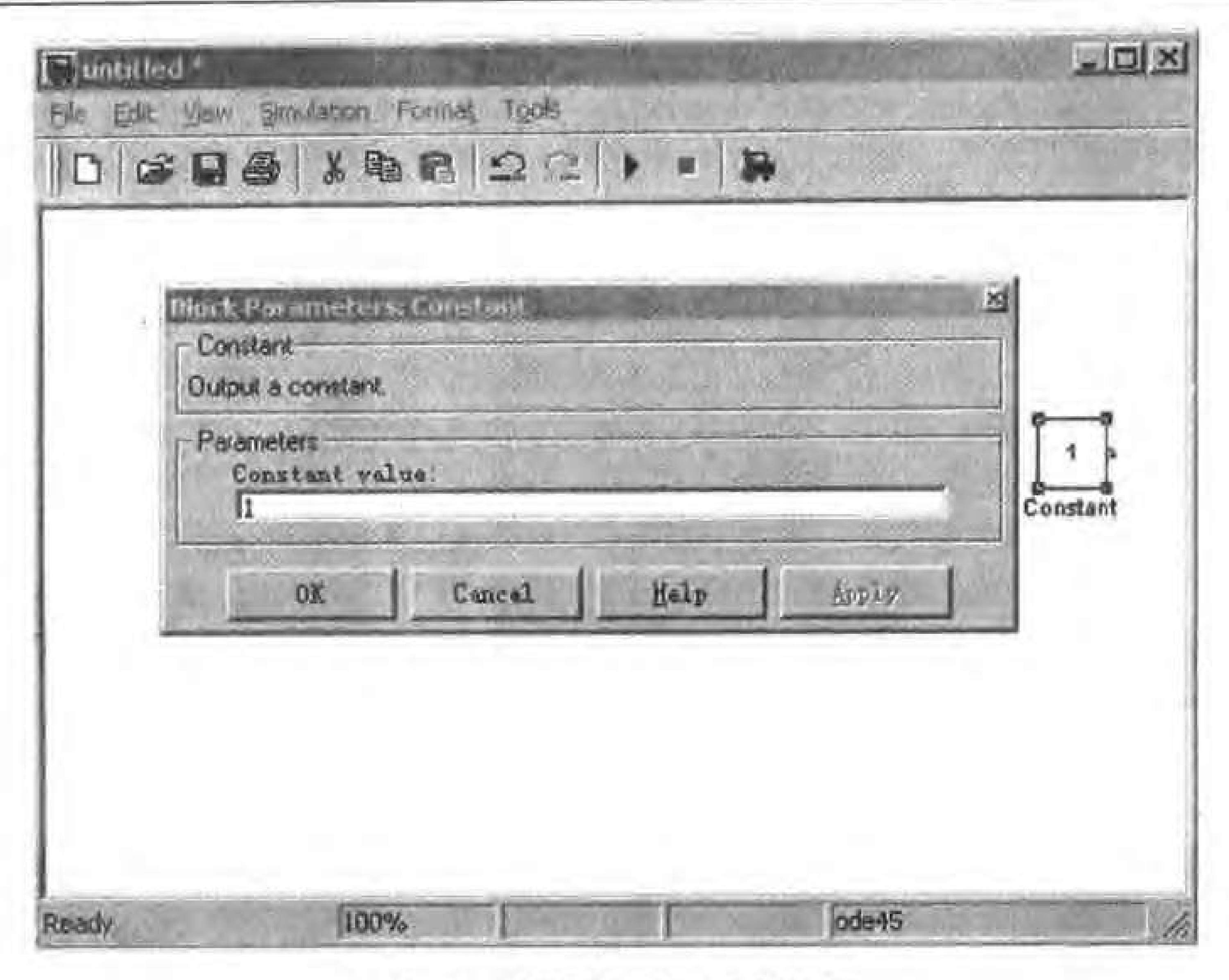


图 4-3　模型窗口及参数设置对话框

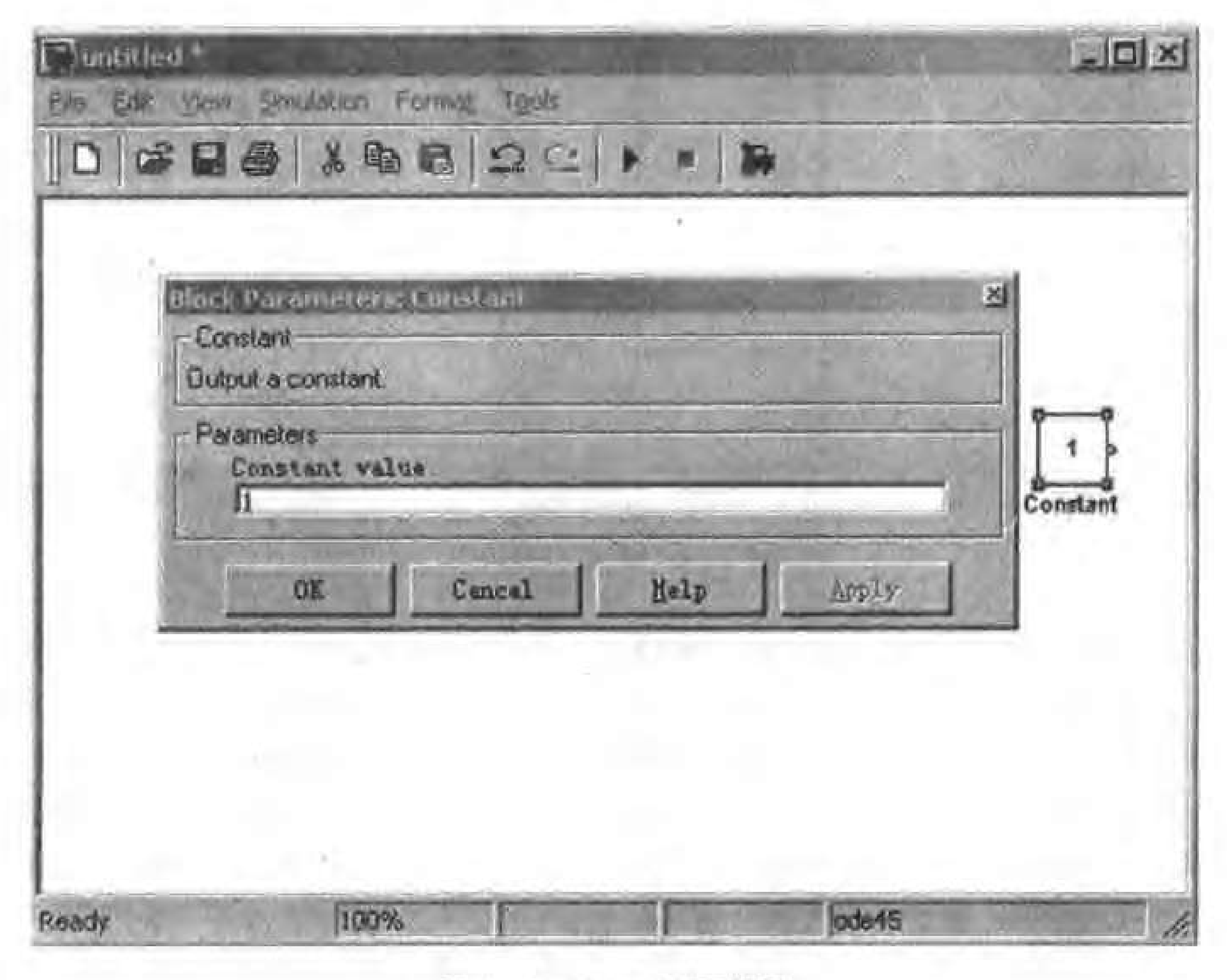


图 4-4　thermo 演示模型

执行该演示程序。执行时，在窗口下面的状态栏会显示出执行状况的进度。

然后，请双击 thermo 中名称为 thermo plots 的模块，这个模块其实就是 sink 子库里的 scope 模块。这样就可以看到仿真的结果了——室内温度、室外温度和热量曲线。在 Simulink 里提供了许多类似于 scope 的可视化显示模块，而这种和实际示波器输出相似的图形化显示结果功能，正是 Simulink 的一个重要的特点：

Simulink 所建立的模型的第二个特点是层次性。请读者随意双击用户界面上显示的模块。有些模块会显示出前面提过的模块参数设置对话框，然而有些模块则会出现如图 4-5 所示的窗口，它就是双击名称为 house 的模块出现的情景。这个图示表示 house 是由图右边所示的一些模块连接而成的。像 house 这种由几个相互关联的模块组合而成的模块在 Simulink 里称为子系统，建立子系统的方法将在后面章节中介绍。而这种一个模块又由许多模块组成的特性，正是 Simulink 的层次性。为了和子系统相区别，这里把模型本身称为模型的顶层系统。层次性的好处是所建立的模型在结构上显得非常清晰整齐，让人一目了然。更重要的是，这个特性使得用户可以选择是采用从上至下建模，还是采用从下至上建模。因此，读者在建模时一定要有层次，不要把所有的模块一股脑地摆在一个模型窗口的顶层，最后乱得连自己都搞不清楚。

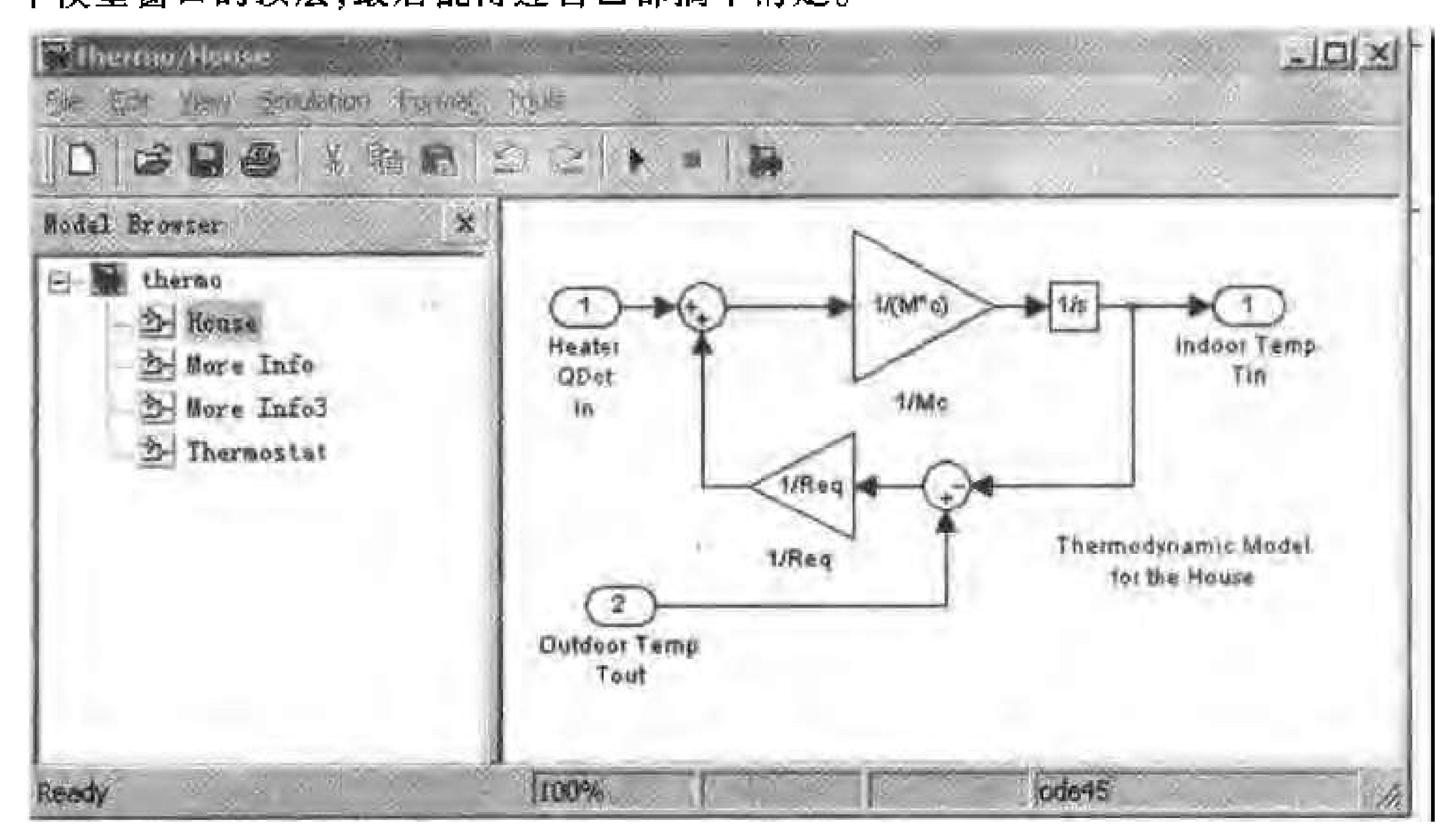


图 4-5 模型的层次性

在 Simulink 里，模拟浏览器可以用来观看一个模型的层次结构。在图 4-5 的左侧，模型浏览器里的树型结构表示了 thermo 模型的层次结构。在最高层，即系统（请用鼠标点击 thermo），这时右边的子窗口就显示系统的连接情况。在 thermo 节点下的子节点，列出了系统层中所有的子系统，同时可以点击这些子节点观看它们的内部结构。Thermo 是一个两层结构的模型，在更复杂的模型里，有可能会出现了系统内嵌子系统的情况。总之，随着层次的下降，模型的细节将会越来越复杂。

第三个要提到的特点是，Simulink 为用户提供了一种封装子系统的功能，用户自定义该子系统的图标和设置参数对话框。还是以 house 子系统为例，它的房屋形状的图标就是封装后的结果，本书的后面章节将会对之进行详细的介绍。

除了上面提到的几个主要的特点要仔细体会，读者还可以做很多其他的尝试，比如观察参数的改变对模型仿真结果的影响。

(1)Simulink 的 scope 模块可以提供一个或多个区域来显示信号的波形，而且用户可以改变每个信号的显示范围，放大或缩小信号的某一个局部，以及设置 scope 的其他属性。读者可以按图 4-6 所示，对 thermo plots 的坐标放大或缩小，或者是改变它的属性(见设置属性按钮)，看看显示的图形会有什么变化。

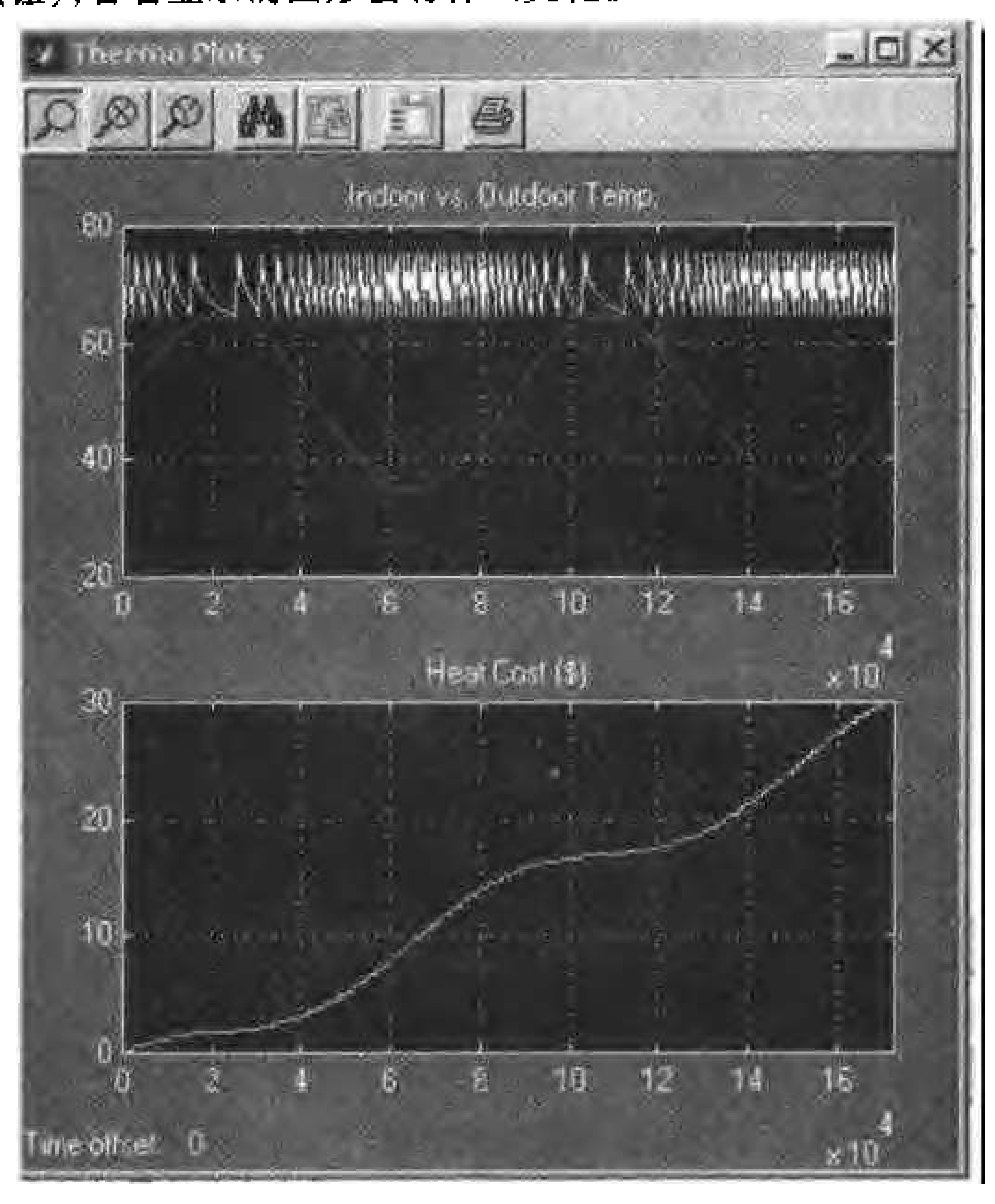


图 4-6　Scope 示意图

(2)改变标签为 set point 的常数模块的常数值参数，比如从 70 变到 80，看看仿真的结果有什么变化。这个模块的作用是设置预期的室内温度。读者可以在仿真运行时，改变这个参数，看看仿真结果又有什么变化。

(3)标签为 daily temperature variation 的正弦模块的作用是设置日温度的变化，试着改变它的幅度值参数，看看仿真的结果有什么变化。

除了 thermo 外，Simulink 还有许多其他的演示模块，读者可以直接在 MATLAB 命令窗口输入 demo，然后在列表框里找 Simulink 选项来获得。对于使用 MATLAB 5.3 的用户，可以使用另外一种方式是在命令窗口输入：

```
>>Simulink3
```

这时出现的 Simulink 模块库窗口是图标形式的，它和前面提到过的库浏览器的区别就像我的电脑和资源管理器的区别，为了和库浏览器相区分，我们把它称为模块库窗口，

简称为模块库。至于选用哪一种界面,则要看个人喜好了,它们在功能上区别不大一样。图 4-7 是模块库窗口的示意图。

本书倾向于使用模块库窗口,因为它里面的模块都用图标来表示,这样比较形象,而且比如输入、输出端口的部分信息也能很直观的看出。

但是在 MATLAB6.0 里,在命令窗口输入 Simulink4.0 则是一个错误的操作,因为 Simulink4.0 的库浏览器已经把这种图标形式的库窗口集成进来了,浏览器的左侧是浏览模块库层次的树形节点图,而当用户选中某个子库的节点时,浏览器窗口的右侧就会显示该子库内包含的所有模块的图标。

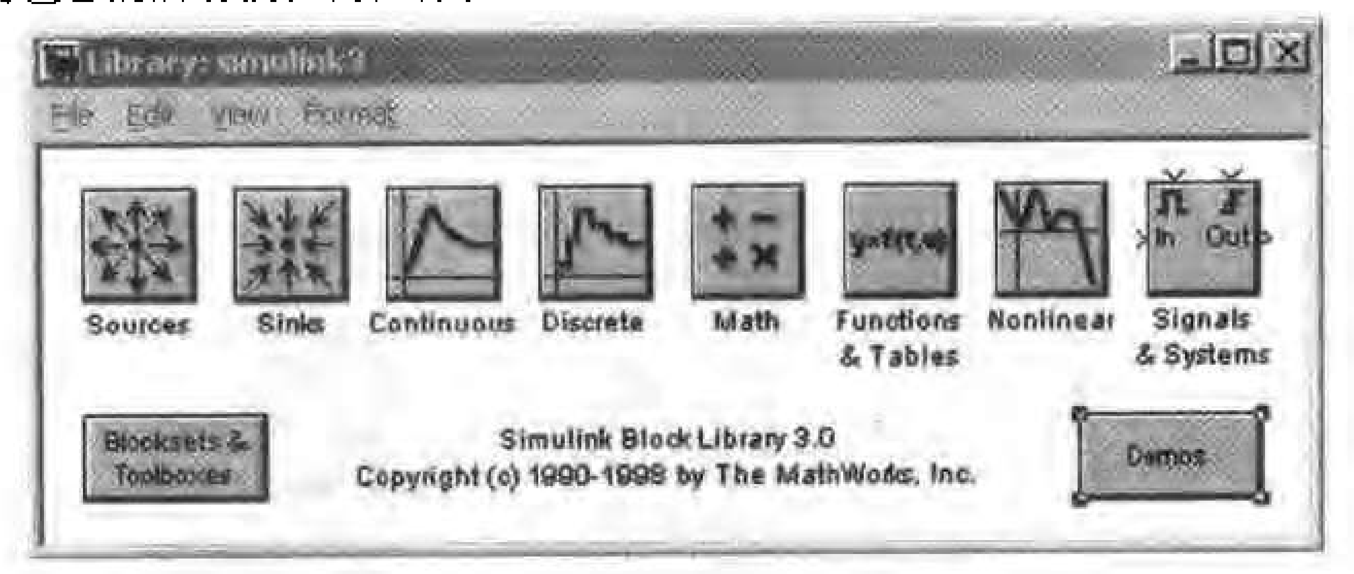


图 4-7　模块库窗口

4.2　Simulink 入门

这一节演示如何用 Simulink 创建一个简单的模型,其中的许多细节在前面一节已有提及,这里主要是把创建模型的整个流程梳理清楚。对于那些通过自己的摸索,已经对这个过程已有所了解的读者,建议最好还是粗略的看一下,然后再学习后面的章节。

将要创建的示例模型所完成的功能是对一个正弦信号进行积分,并显示积分的结果。图 4-8 是最终的模型的样子。

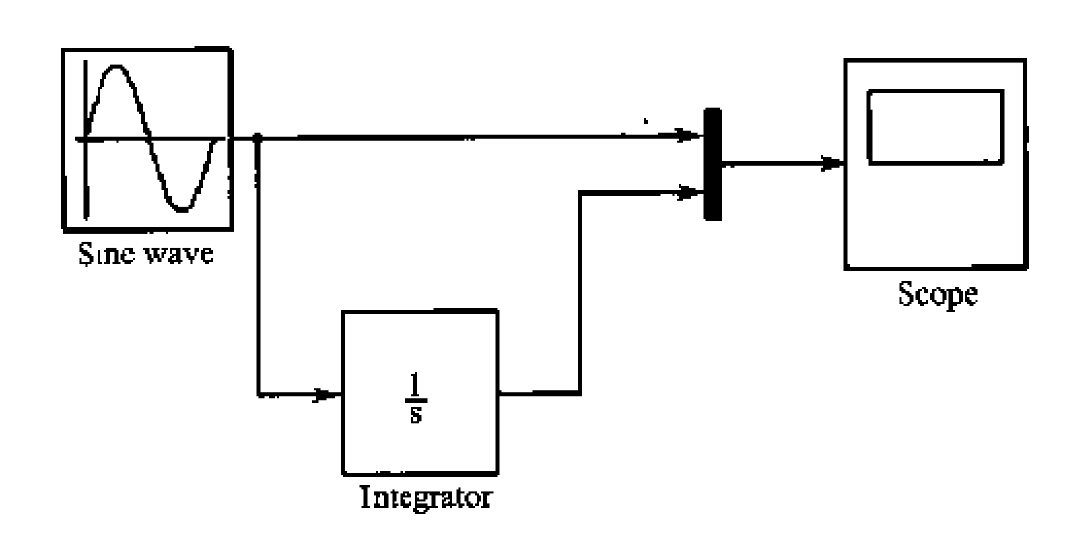


图 4-8　建好的模型

一般来讲,在模型结构都已经设计好的基础上,用 Simulink 建立模型的过程可以简单概括为:在 Simulink 的模块库里找到你所需要的模块,并把它们拖到模型窗口中,将这

些模块排列好，然后用直线把各个模块连接起来。具体的操作步骤为：

(1) 启动 Simulink 模块库浏览器窗口（这当然是必须的）。

(2) 新建一个空白模型，为此请点击库浏览器工具栏上的空白按钮（如图 4-9）。在 Simulink 里，模型是保存在模型文件里的，新建一个空白模型，也就是新建一个空白的模型文件，模型文件的后缀名为 .mdl。也可以在模型窗口新建一个空白模型，其操作是使用 File 菜单下的 new a model 命令。

(3)在模型库浏览器窗口中找到所需的模块。此模块包括的模块有：正弦波发生器、积分器、复合器和示波器。它们的位置分别是：正弦波发生器在 source 子库，积分器在连续系统子库（见图 4-9），复合器在信号和系统子库，示波器在接收子库。

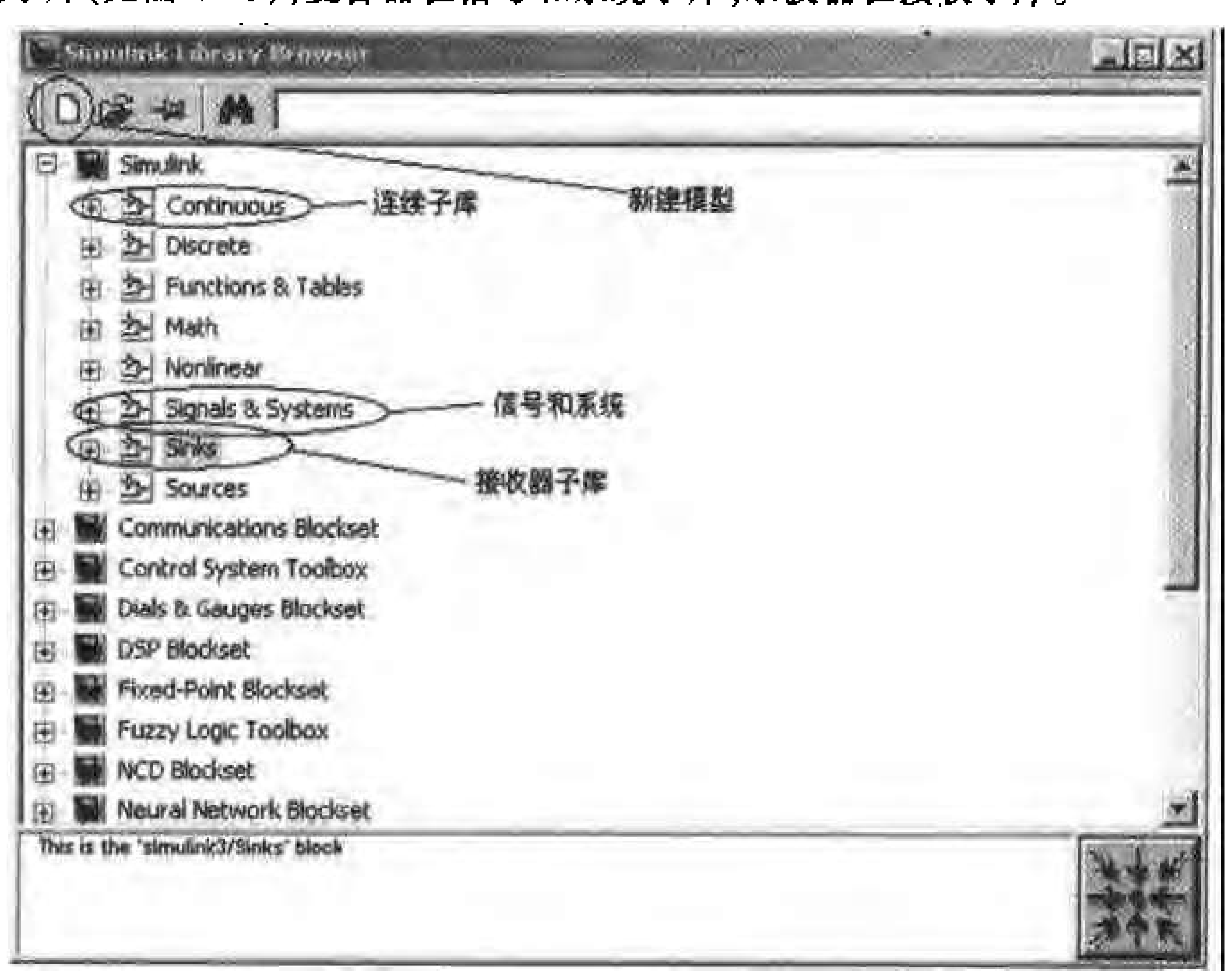


图 4-9 从模块库查找积分所需的模块

(4)分别将所需的各个模块从库里拖到空白的模型窗口。这时，Simulink 会在模型窗口复制出这些模块。复制后的结果如图 4-10 所示。

(5)将这些用户界面中的模块排列好，并按图 4-8 把它们用直线连接起来，注意模块的输入只能和模块的输出端相连接，示意图见 4-10。

在图 4-8 所示的模型里，大部分的连线都是从输入端口到输出端口，只有一根直线从一条输出线到 Mux 牛块的输入端口，这种连线称为分支线。它表示一个模块的输出同时作为多个模块的输入。它的操作也很简单：

① 首先，把鼠标定位在 Sine Wave 和 Mux 的连线上；

② 然后，按下 Ctrl 键并保持，同时按下鼠标左键，并拖动鼠标至 integrauor 的输入端（拖动时不要松开 Ctrl 键），然后再松开左键和 Ctrl 键。图 4-11 描述了这个过程。

把所有的连线都连好之后，这个仿真模型基本上就建好了。

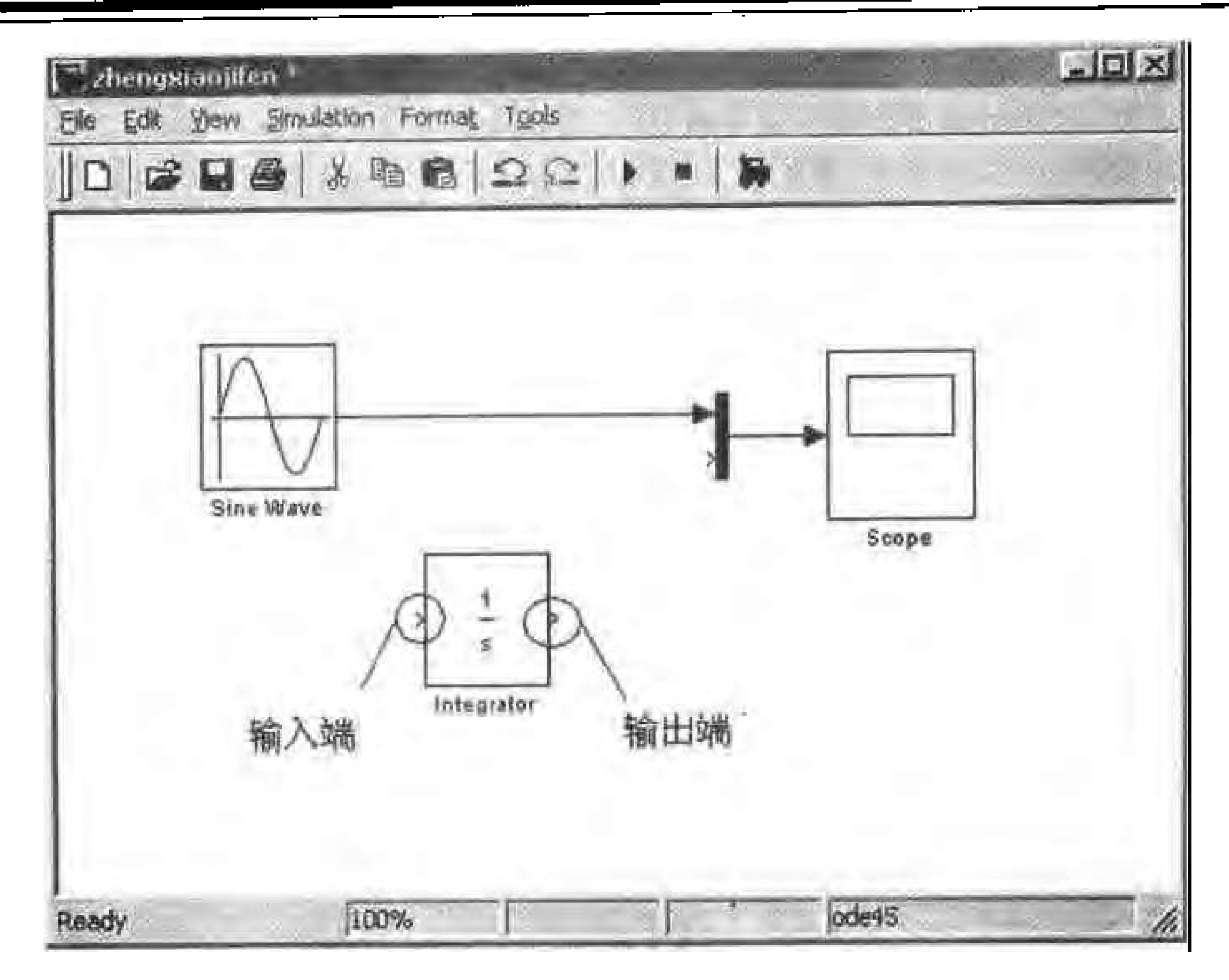


图 4-10　连线示意图

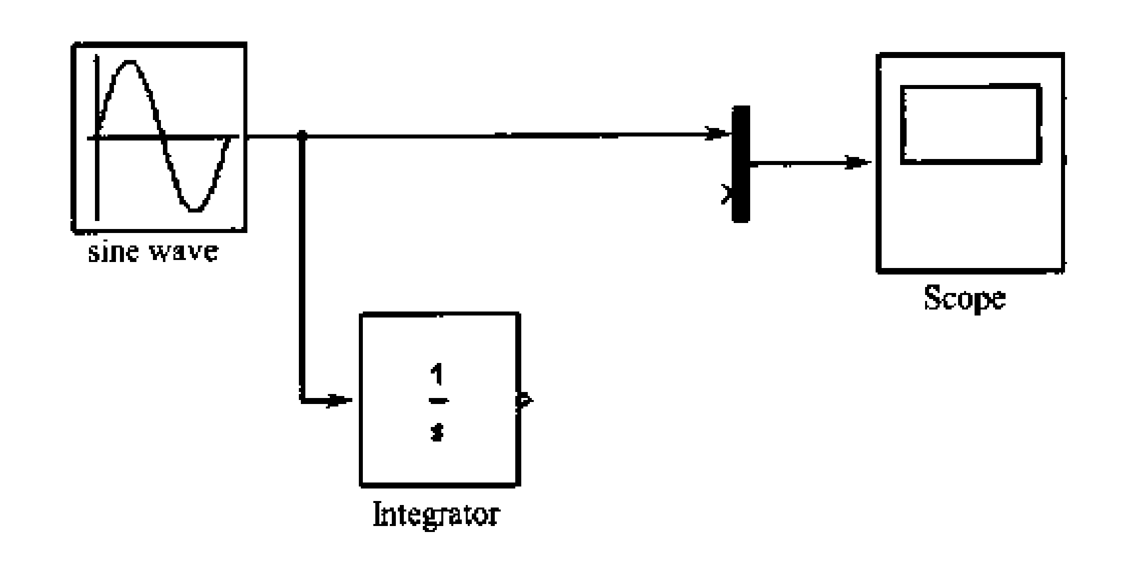


图 4-11　建立分枝连线示意图

注意：用户界面里不能有孤立模块存在，即不能有和别的模块没有任何连接的模块。模块的每个输入端，都要为它指定输入信号，即都要有连线。但是输出端可以空置。

现在，双击 Scope 模块打开 Scope 窗口，以观察仿真的输出波形。在开始仿真之前，先把仿真的时间设置成 10 s。具体的操作为：首先，从 Simulink 菜单里选择 parameter 项，打开设置仿真的参数的对话框，见图 4-12。注意到 Solver 页的 Stop time 的值为 10.0，这个参数规定了仿真的结束时刻，它的缺省值就是 10.0。按 OK 按钮关闭对话框，所作的改变就生效了。和 Simulink3.0 的仿真参数对话框相比，Simulink4.0 增加了一个 advance 页。关于仿真参数对话框的详细说明，在后面章节中会介绍。

运行仿真模型，就可以在 Scope 里观察积分输出的波形，如图 4-13 所示。在 Scope

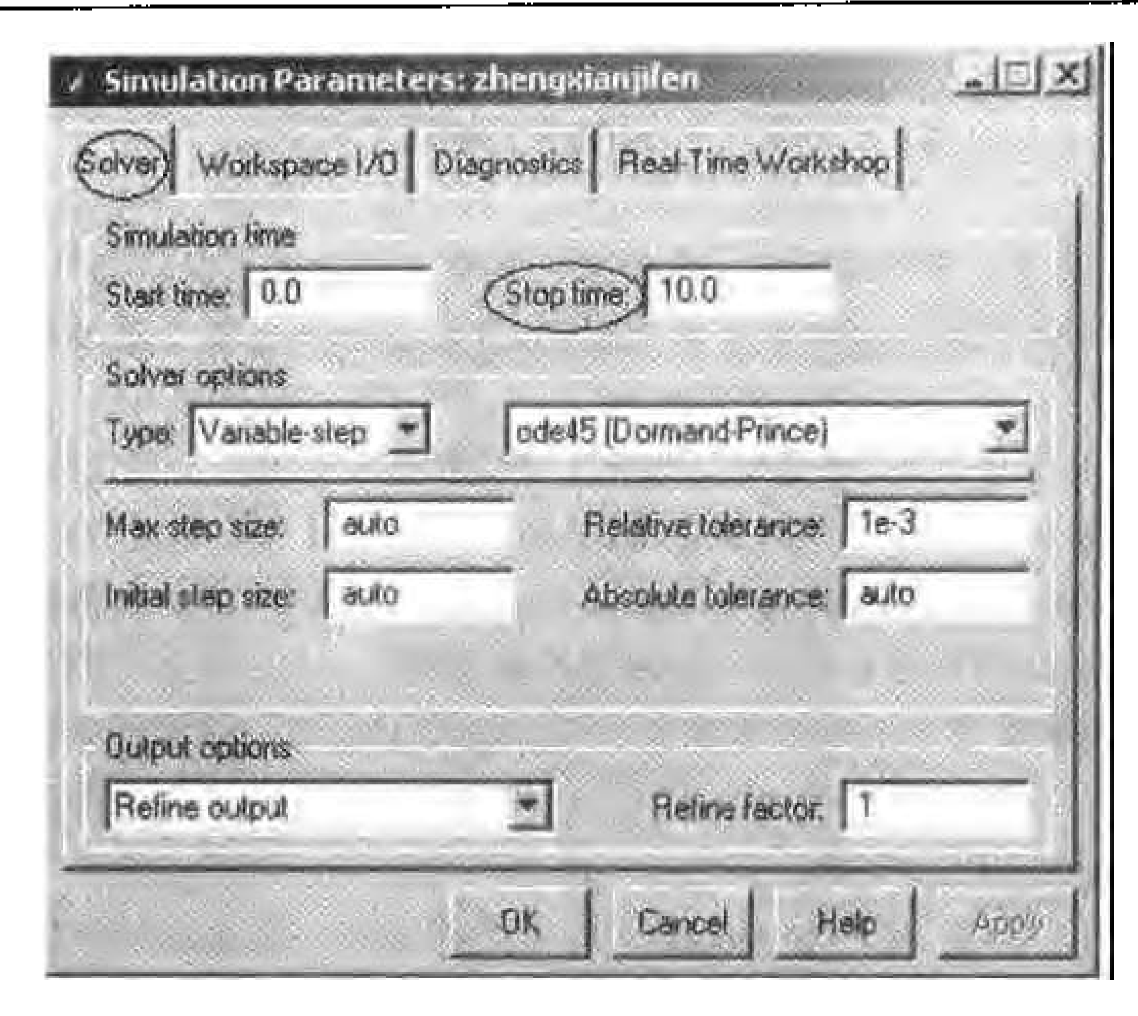


图 4-12　仿真参数设置对话框

的显示区域有两条曲线，一条是积分输出的曲线，另一条是输入的正弦曲线。

最后，选择 File 菜单下的 save 命令保存模型，模型文件后缀名为 .mdl。

至此，用 Simulink 建立模型的基本操作就介绍完了。

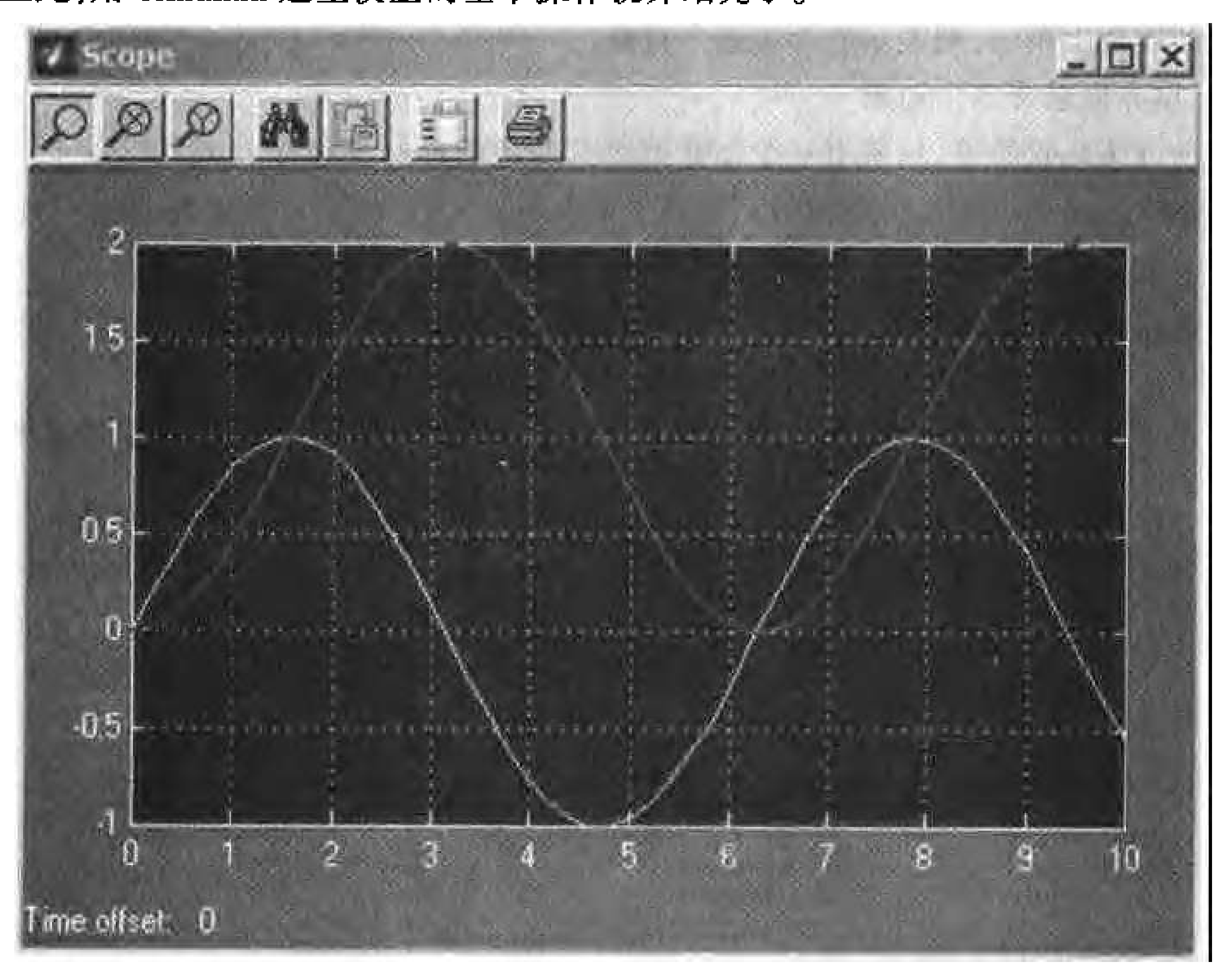


图 4-13　仿真结果的输出曲线

4.3 熟悉 Simulink 模型窗口

用 Simulink 建立模型的操作除了将模型从库中复制到模型窗口之外，大多是在模型窗口完成的。读者要想熟练地掌握这些操作，详细地了解其中各个菜单和按钮的功能是十分必要的。

让我们从 File 菜单开始，模型窗口的 File 菜单和通常的 File 菜单没有什么大的区别，包括：new、open 和 save 等等常用的菜单项，分别用来新建立一个模型、打开已有的模型以及保存当前模型等。需要注意的是 model properties 项，它的作用是设置模型的一些维护属性，比如模型的版本号、建立时间以及模型的描述等等。另一个要注意的命令是 Source control 命令，这用来设置 Simulink 和源控制系统(SCS)的接口，所谓的原控制系统是管理文件的一种工具。而 Simulink 和 SCS 的接口，使得用户可以将模型文件从 Simulink 登记到 SCS 中，或者是从 SCS 中注销。

接下去的 Edit 菜单汇集了 Simulink 一些重要的操作。前面的一些项是基本的编辑命令：undo、copy、cut 和 past，它们的作用和其他程序里的相同。在进行任何操作前，必须先选定操作的对象。在 Simulink 里，选定一个对象，只需要用鼠标左键单击该对象即可。若要选取多个对象，可以使用两种不同的方法。

(1)逐个选择法

选择时，按住 Shift 键，然后再用鼠标左键逐个单击待选定的对象即可，如果在一个已被选中的对象上单击，则表示放弃了这个对象。

(2)用方框选择多个对象

这种方法比较简单，只要用一个方框把所有的待选择对象包含起来就可以了。定义方框的操作很简单，首先，把鼠标指针移到一个空白处，然后按下鼠标左键，就定义了方框的起始位置，然后拖动鼠标，把指针移到任意一个你想到的位置，就可以看到一个虚线框，在虚线框中的对象表示都被选中了。放开鼠标左键，图 4-14 中的图表示了这个过程，图 4-15 表示了这个动作后的结果是多个对象被选中了。

读者可以自已尝试这些命令的使用方法，例如看看如何在模型内或模型之间拷贝模

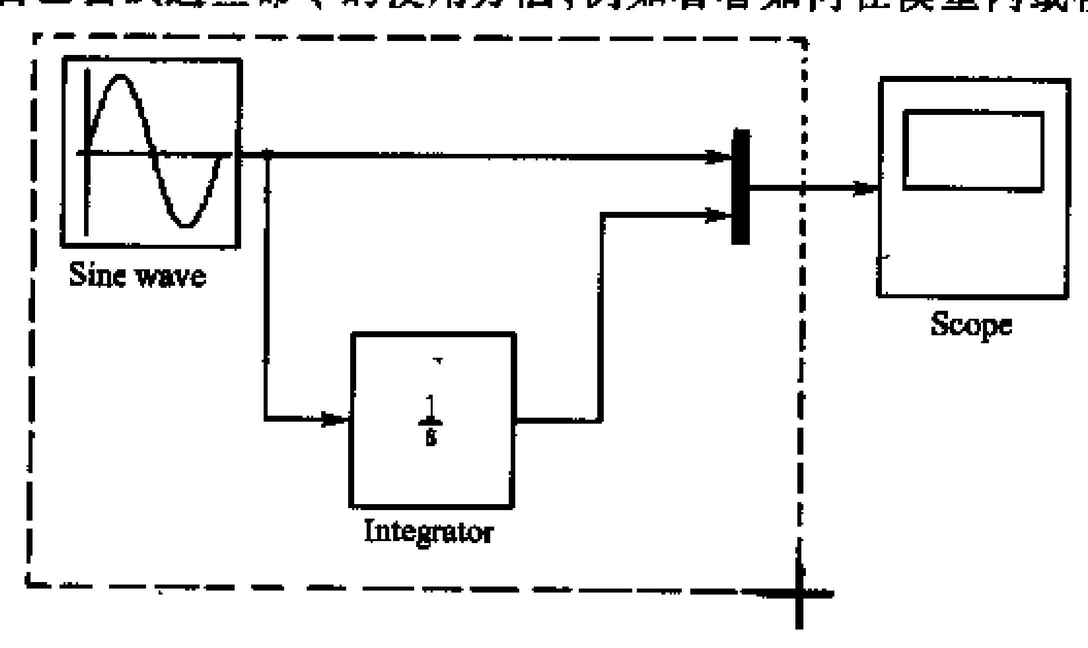


图 4-14 用鼠标框选多个对象

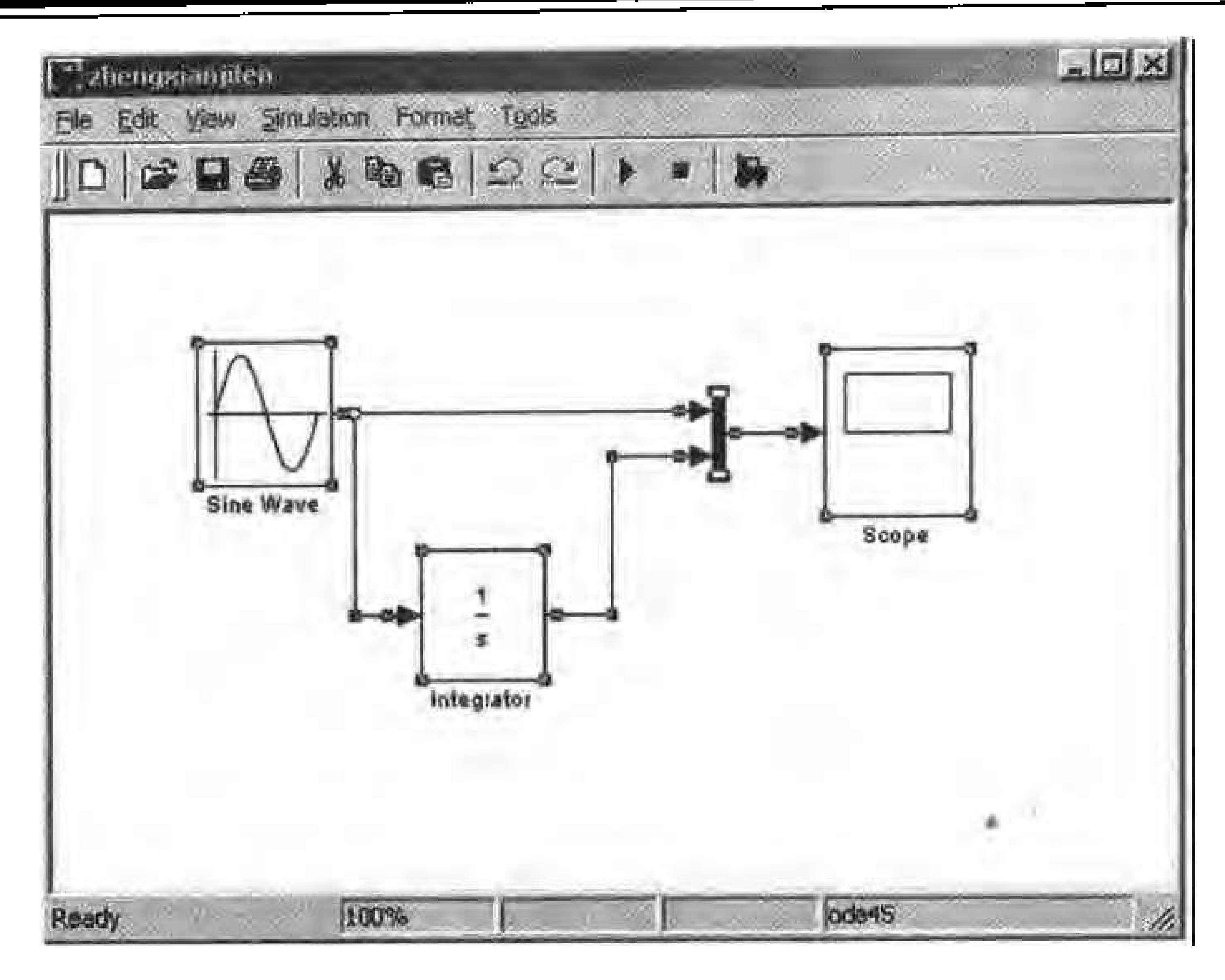


图 4-15 多个对象被选中后的情况

型,如何移动模块等等。提醒一点,在进行这些操作之间最好先将模型另存一下。

Edit 菜单里的 copy model 命令的作用则是把当前的整个模型当成图片来拷贝下来,作为其他的用途。比如,本书中关于模型的许多插图都是用这个命令而得到的。

Edit 菜单接下去的几个命令就比较重要了。在一般情况下,它们都处在不被激活的状态(为灰色)。这是因为不同的菜单命令有不同的使用对象。例如,signal properties 项是用来设置信号属性的,如果你当前的选中对象是一个模块,那么这个菜单命令仍然处在不可用的状态。而它下面的 model property 却变成了可用的状态。在 Simulink 里信号是和模块的端口以及与端口相连的直线相关的,因此只有选中了一条连线,signal property 才会变成可用。单击该命令出现的信号属性设置对话框如图 4-16 所示。

其中 signal name 属性的作用是标明信号的名称,设置这个名称反映在模型上的直接效果就是与该信号有关的端口相连的所有直线附近都会出现写有信号名称的标签。图 4-17反映了它的效果。这里,设置给 Sine Wave 模块输出信号的名称为 sine。

Block discription 指该信号的描述信息,比如功能、类型等,建模者可以在上面写入有助于理解模型的任何说明信息。在建模时,养成一个良好的注释习惯是十分重要的,无论是对他人还是对建模者自己都是极有好处的。

document link 用户可以在里面输入一个指明与该信号有关的文档的 MATLAB 表达式,例如你可以输入 web 地址等等。

信号属性对话框上的后面两个属性只有在使用 RTW 或者生成 C 代码时才会用到,本书的后面将会讲到它的用法,用户在此可以先把它忽略掉。

单击 Edit 菜单下的 model propeties 命令可以设置模块的属性,首先会打开一个模块

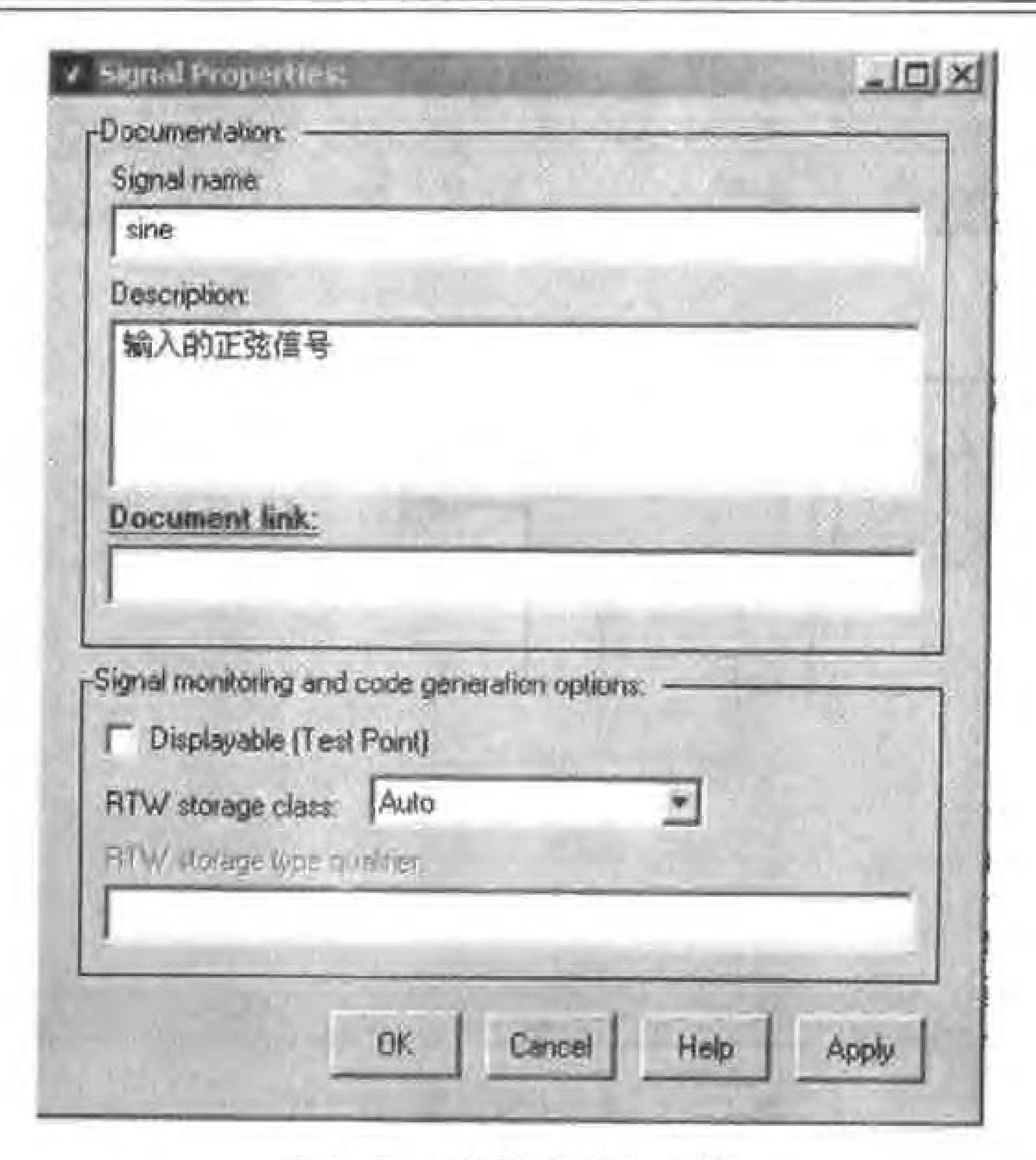


图 4-16　信号属性设置对话框

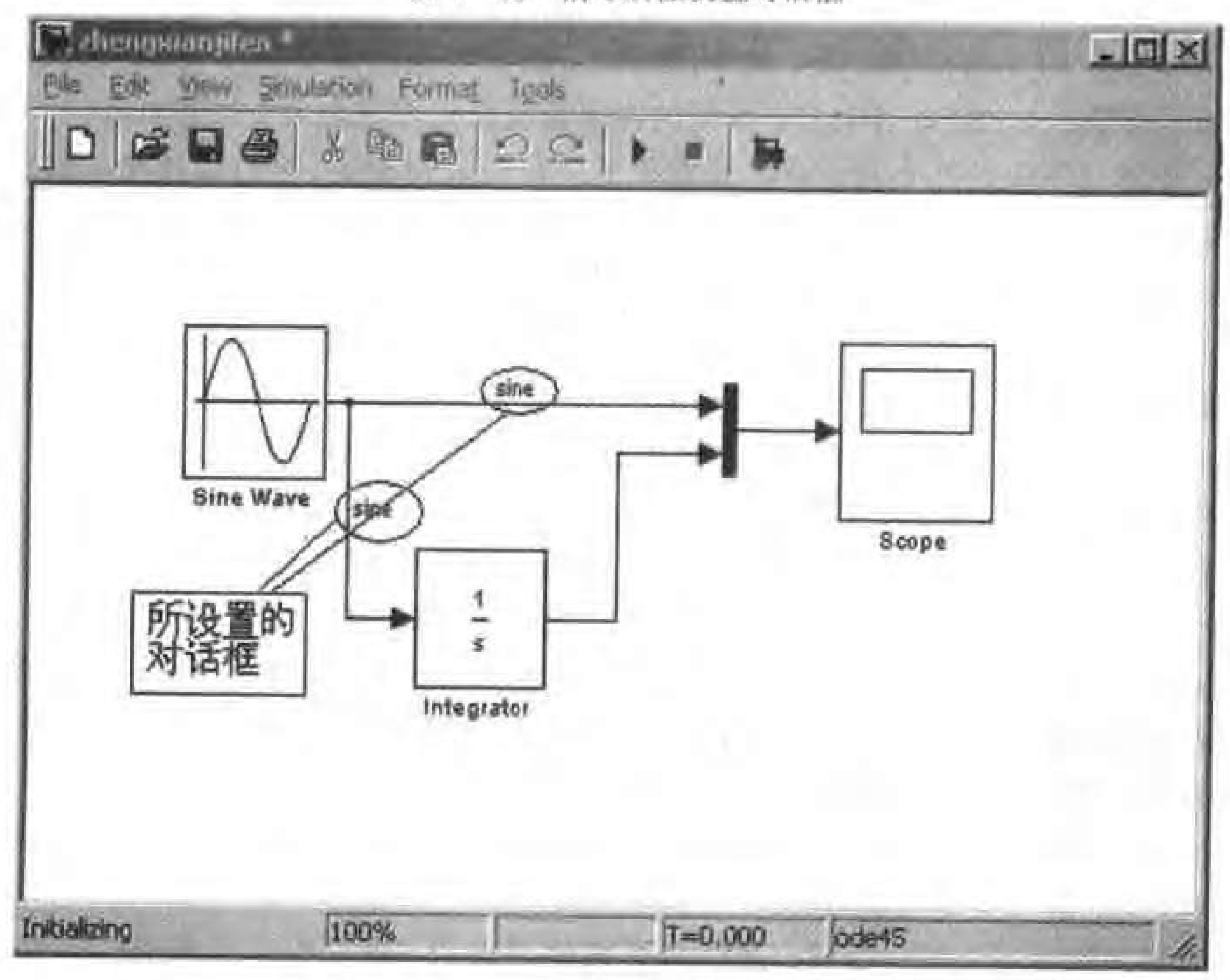


图 4-17　信号名称设置后的结果

属性设置对话框，见图 4－18。

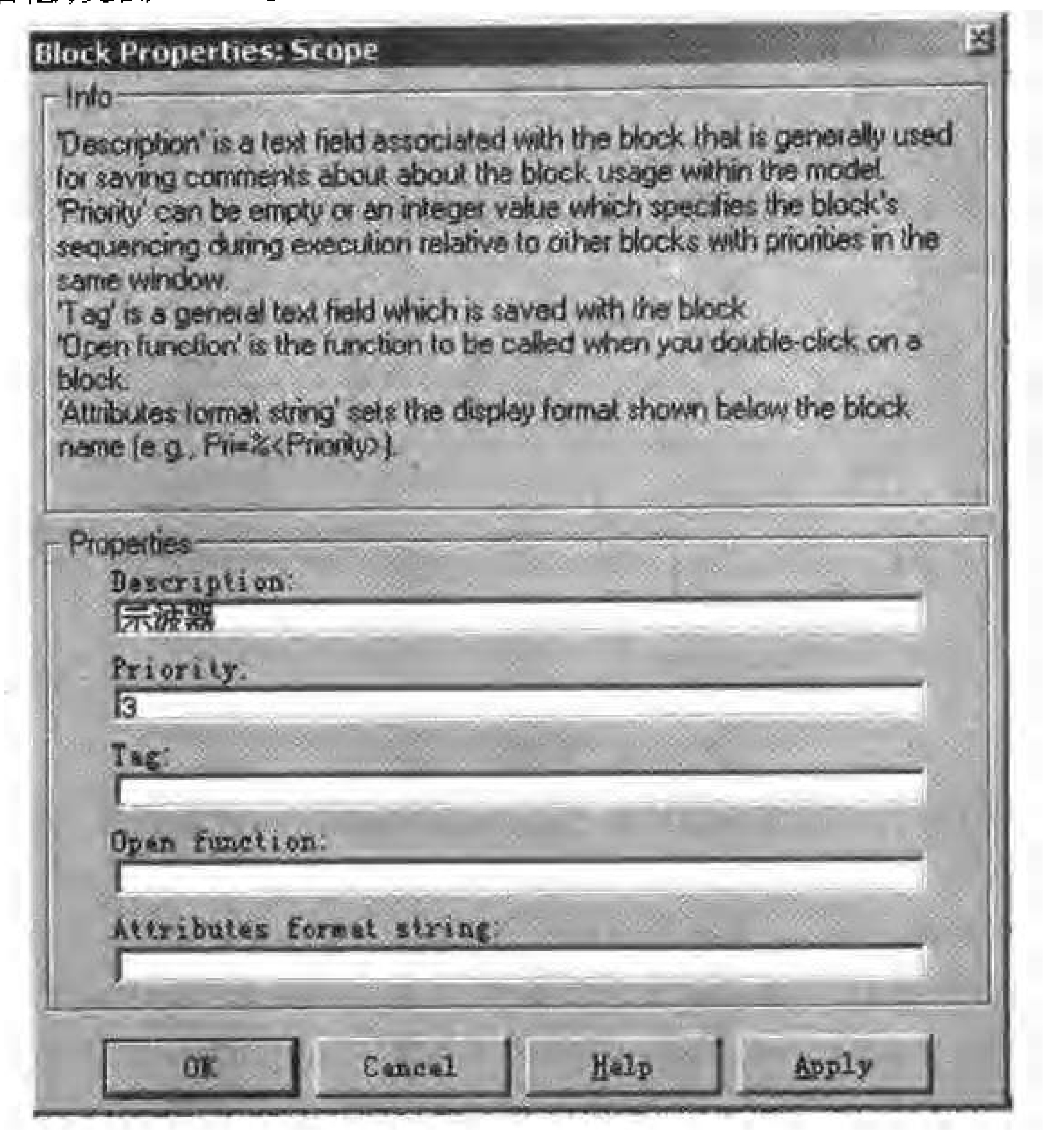


图 4－18　模块属性对话框

图 4－18 中各个属性的作用在对话框上都有很详细的说明，这里只稍稍说明一下。对于 description 属性，给它设置一个值如图中的“示波器”，要进行一些设置才能看出模型的变化。这里有两种方法，一种方法是按图 4－18 所示，在 Attributes format string 里输入：

pri = % < priority > ; des = % < description > ，按 OK 按钮之后，模型图就呈现如图 4－19所示的样子。

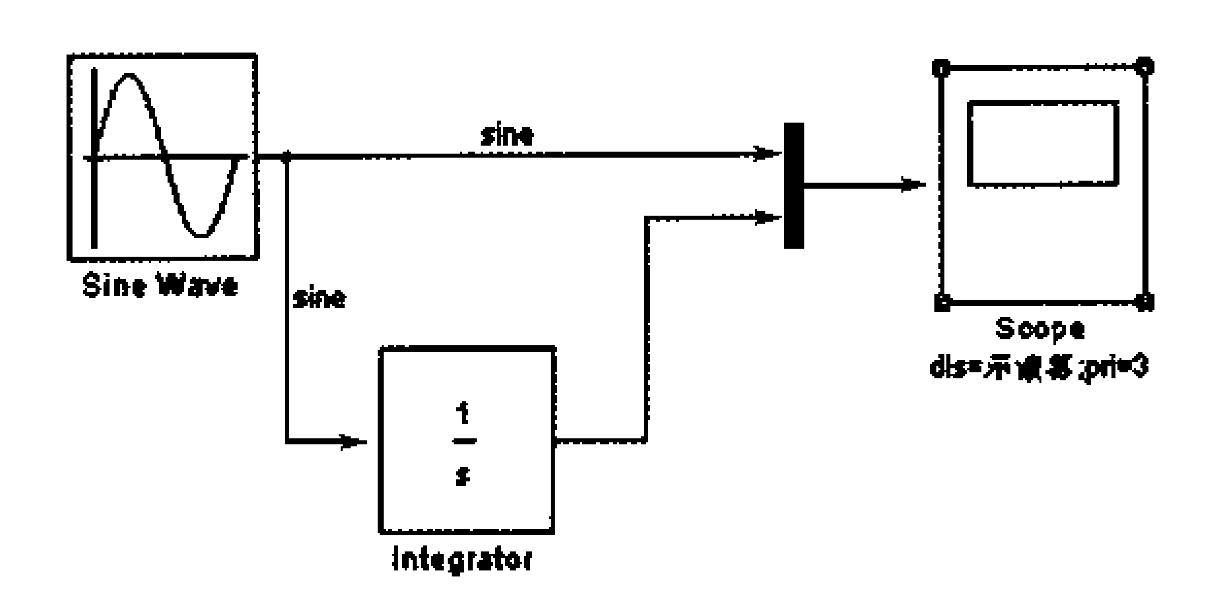


图 4－19　设置属性格式之后的模型图

读者可以类似地把模块其他属性值标注到模块的下面。

第二种方法是,在模块的数据提示里显示用户模块的描述字符串。所谓模块的数据提示,就是指当指针在模块上停留时,会出现提示信息。需要做的设置是,把View的Block Data Tips的Show Data Tips和下面的User de都设置为被选中状态。于是,当鼠标指针在Scope上停留时使出现如图4-20所示的提示信息。

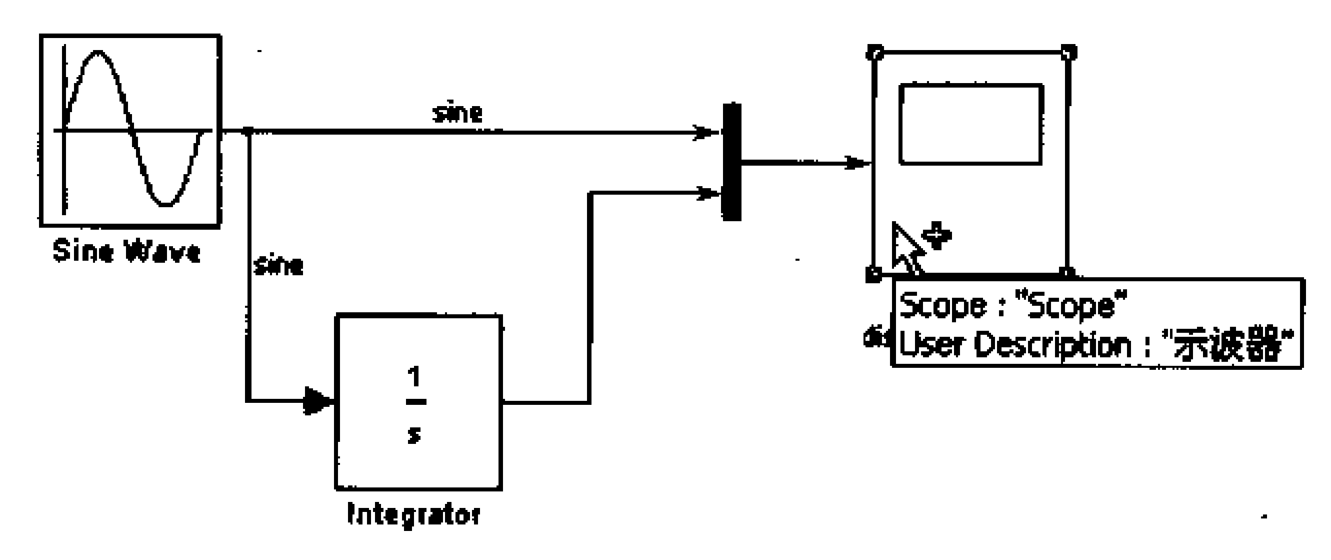


图4-20 数据提示的示意图

Priorty属性的作用是设置该模块在Simulink开始执行仿真模块时,相对其他模块执行的优先级。

Open function属性是一个很有用的属性,它说明了在模块被双击后,Simulink要调用的函数,这种函数在MATLAB里称为回调函数,关于它的概念将在后面的章节中介绍。

接下去的Create Subsystem的作用是建立子系统。如何创建子系统以及如何封装子系统等内容在下一章中介绍。

从Goto Library Link一直到unlock library是关于库的一些操作,放在后面章节中介绍。最后的Update Datagram作用是更新模型图表。它的作用是能方便地把最近对库里的源模块(模型的模块都是库里模块的一个拷贝)的任何改动,更新到现有的模型中,而不需要用户手动更新。关于它的详细说明也在后面章节中介绍。

下面来看看View菜单。其中大多数的命令项读者都可以自己尝试它的作用。比较费解的一项是Block Data Tips,其子菜单中的Show Block Data Tips一项选中和不选中的区别是当你把鼠标指针移到某一模块时有或没有模块的数据提示信息。而Show Block Data Tips下面的那些选项则说明提示信息里应该包含的信息。图4-21展示了Show

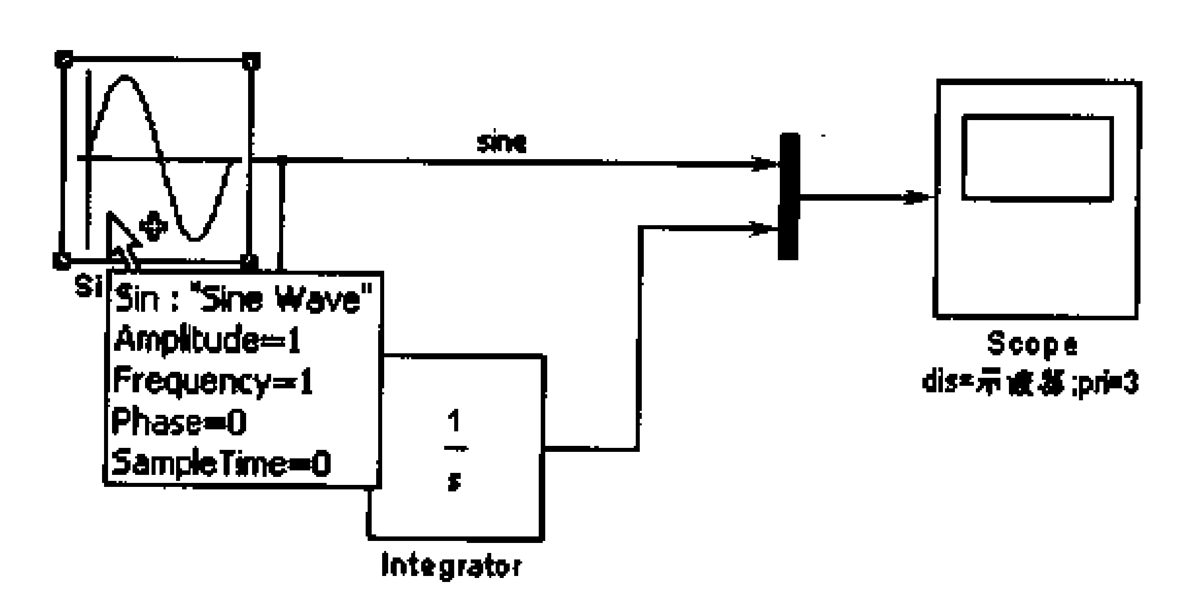


图4-21 模块数据提示

Block Data Tips 被选中后，含有模块的数据提示信息的样子。

Simulink 菜单下的 Start 命令的作用是开始运行仿真。Stop 则是终止仿真的运行，不管是否到达仿真参数里设置的终止时间。而 Parameter 项，则是用来设置参数的。最后面的两项，是相互排斥的选项，它们表示 Simulink 的两种不同的工作模式——正常模式和外部模式。请读者选择 Normal 模式，至于外部模式将在后面章节中介绍。

Format 菜单下的命令的作用大多数都可以试出来。这里只强调最底下的几个选择项。图 4-22 就是所有的选择项都选中时，前面示例模型的样子。Simulink 里，几乎所有模块都支持标量或者向量的输入信号，也能产生标量或者向量的输出信号。例如图中的 Mux 的输出信号就是一个两位宽度的向量信号。要判断图表上哪条直线传递了向量信号，可以选中 Wide Vector Lines 选项，这时模型图表中所有传递向量信号的连线都会变宽，图 4-22 的 Mux 的输出信号就是这种情况。而 Vector Line Widths 选项则在向量化的直线上标出了

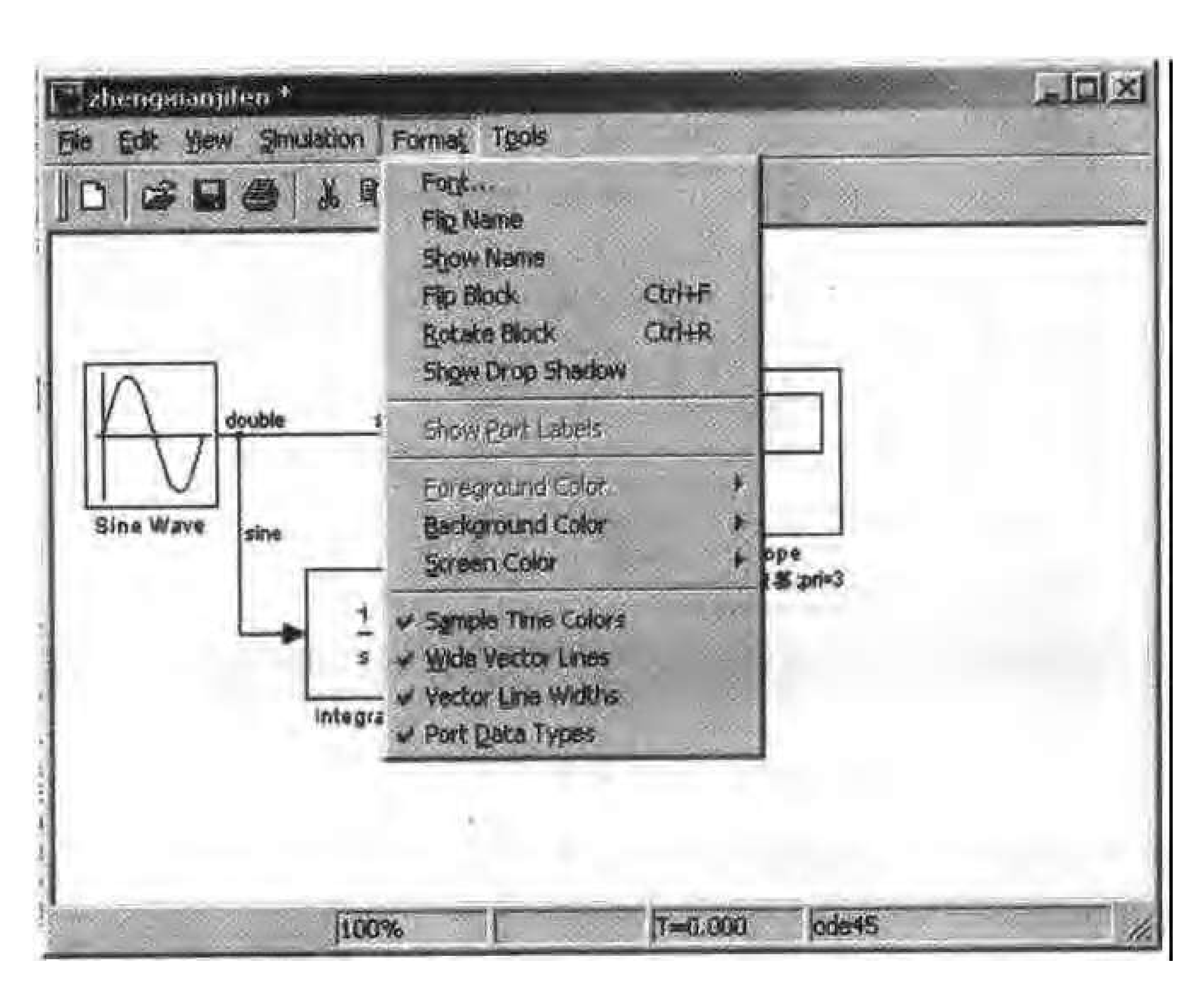


图 4-22 Wide Vector Lines 选项的图示

向量的宽度，Port Data Types 选项的作用是标出模块中的端口的数据类型，如果是向量信号还指出它的宽度。

Sample Time Color 这一个选项的作用是通过模块边框的颜色来反映模块的采样时间快慢。由于在上例中，各个模块的采样时间都相同，所以选中这个选项，对图表外形的作用不明显。读者可以在 Simulink 菜单的 Demos(演示模型)里选择一个模型来观看这个选项对图表外观的影响。关于采样时间的详细信息请看后面章节。

最后还要介绍的是 Tools 菜单，Simulink4.0 这个菜单里的命令和 Simulink3.0 有比较大的变化，Simulink4.0 的许多新增功能都集中在这个菜单里。其中，Simulink debugger 命令打开 Simulink4.0 新增的图形调试工具，而以往的 Simulink 调试功能都是基于命令的形式的。关于如何进行调试将在后面章节中再详细说明。

而 Data Explorer 选项则是打开一个数据管理器，这个工具也是以前的 Simulink 版本所没有的。图 4－23 是在 Simulink 的演示模型 f14 下，它被打开后的样子(打开 f14 的方法是在命令窗口中输入 f14)。数据管理器的作用是管理模型中定义和用到的一些数据变量，显示它们的数据类型，但是无法添加变量或者修改变量的属性。

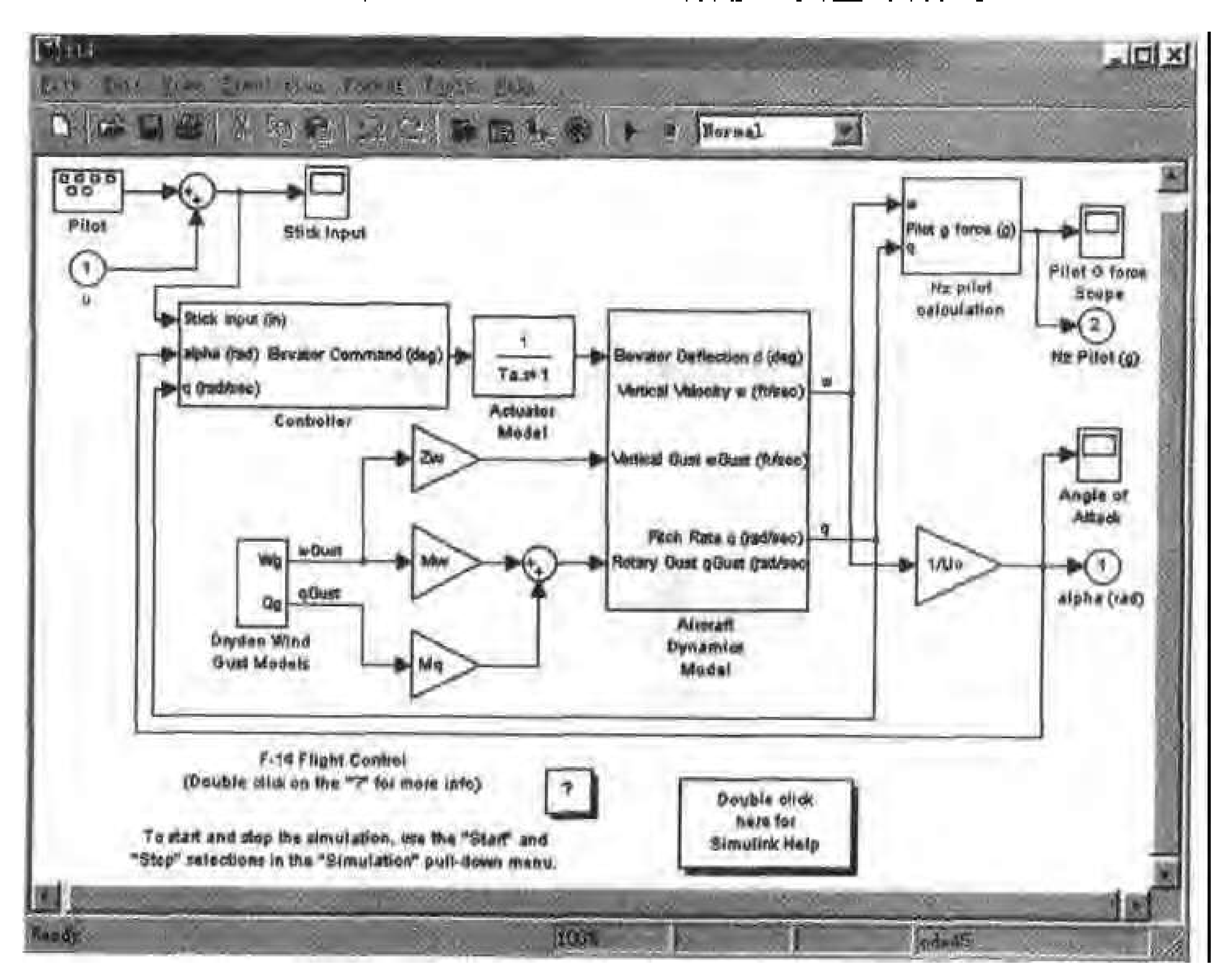


图 4－23 演示模型 f14 被打开的样子

Tools 菜单的最后两项都是和 Real Time Workshop 有关的命令，本书将在后面章节中详细介绍。

4.4 键盘和鼠标的操作

下面的这些表格汇总了 Simulink 里对模块、直线和信号标签进行各种操作的鼠标和键盘的用法。

表 4－1 列出了一下些关于模块的键盘和鼠标操作，这些操作仅在 Microsoft Windows 环境中适用，对于使用 Unix 的用户，请看 Simulink 的用户向导。表 4－2 列出了关于直线的各种操作。

表4-1 对模块进行操作

任　务	Microsoft Windows环境下的操作
选择一个模块	按鼠标左键
选择多个模块	Shift+鼠标左键
从另一个窗口复制模块	拖动模块
移动模块	拖动模块
Duplicate	Ctrl+鼠标左键,并且拖动鼠标,或者是鼠标右键和拖动
在模块间连线	鼠标左键
断开模块间的连线	Shift+拖动

表4-2 对直线进行操作

任　务	Microsoft Windows环境下的操作
选择一条直线	鼠标左键
选择多条直线	Shift+鼠标左键
画分枝直线	Ctrl+拖动直线;鼠标右键并且拖动直线
用方框框上模块	按鼠标左键
移动直线段	拖动直线段
移动顶点	拖动顶点
建立直线段	Shift+拖动直线

对信号进行标注以及在模型图表上建立描述模型功能的注释文字,是一个很好的建模习惯。读者请看下面的例子,来看看什么是信号标签和注释。如图4-24所示。

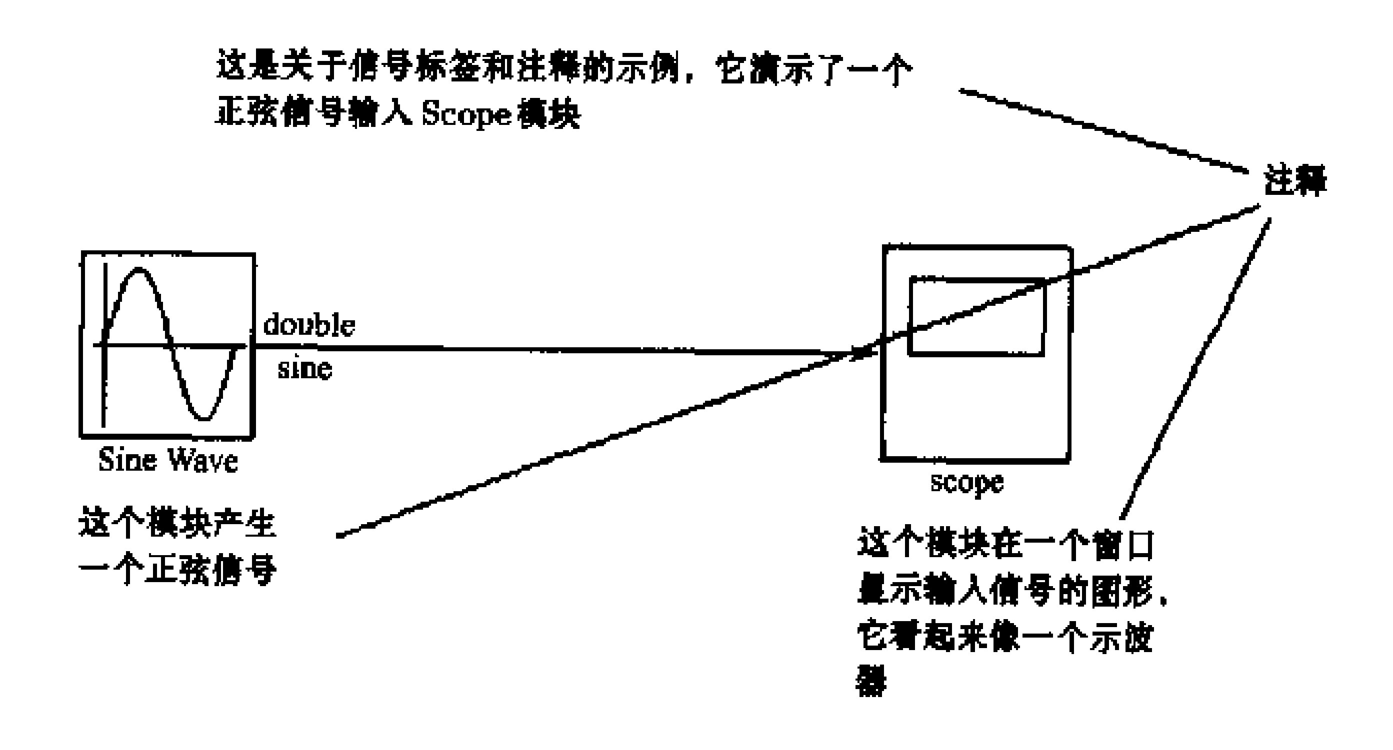


图4-24 注释和信号标签示例

表4-3列出了关于信号标签的一些操作。给一个信号添加上标签,只需要用鼠标左键在直线上双击,然后输入文字就可以了。至于其他的操作,请读者看表4-3即可。

表 4－3 对信号标签进行处理

任　　务	Microsoft Windows 环境下的操作
建立信号标签	在直线上双击，然后输入
复制信号标签	Ctrl＋拖动标签
移动信号标签	拖动标签
编辑信号标签	在标签内单击，然后编辑
删除信号标签	在标签上 Shift＋单击，然后按删除键

建立模型注释的操作也很简单，只要在模型的空白处，用鼠标左键双击，然后输入注释文字即可。详细的命令说明请参阅表 4－4。

表 4－4 对注释进行处理

任　　务	Microsoft Windows 环境下的操作
建立注释	在模型图表里双击，然后输入文字
复制注释	Ctrl＋拖动标签
移动注释	拖动标签
编辑注释	单击文字，然后编辑
删除注释	按 Shift 键＋选中的注释，然后按删除键

4.5 模块库简介

用 Simulink 建立仿真模型的过程，简单地就可以理解为将模块库里的模块拼搭在一起。因此，读者必须了解其中的各个模块的作用，这一节就简单地介绍一下 Simulink 的库浏览器。

库浏览器的作用是让用户快速地对模块进行定位并且将它们复制到模型里去。用户可以像前面所讲述的浏览库树来定位模块，也可以通过查找功能来获得某个模块的位置。关于它们的操作，读者可以查阅前面小节里的内容，即使是自己摸索也是不难获得的。下面想重点说的是关于各个子库以及各个模块的功能。

库浏览器中模块的多少，或者根节点的多少，取决于用户的安装。但至少会包括 Simulink 节点和 Simulink extras 节点。读者点击（单击节点的加号，或者是双击名称）Simulink 节点，可以看到它包含下面这些子节点。

(1)Sources 库，它包含了产生信号的模块，如 Sine wave（正弦波）和 Random Number（随机数发生器）等。

(2)Sinks 库，它包含的模块用于显示或者写模块的输出。如经常用到的 Scope 模块，这个库里的模块还可以把输出写入文件中或者是保存变量到 MATLAB 工作空间。

(3)Discrete 库，它包含了描述离散时间系统组件的模块。其中最典型的是 Discrete Transfer Fcn 模块（实现离散传递函数）、Discrete Filter 模块（实现 IIR 和 FIT 滤波器）等。

(4)Continuous 库，包含了描述线性函数的模块。典型的代表有 Derivative 模块（输入

输出信号的微分)、Integrator模块(对输入信号进行积分)、State - Space模块(实现线性状态空间系统)等。

(5)Nonlinear库,包含了模块描述非线性函数。例如Quantizer模块(将一个信号按一定间隔进行离散化)、Switch模块(在两个信号间切换)。

(6)Math库,包含了描述一般数学函数的模块。其中的模块的功能就是将输入信号按照模块所描述的数学运算函数计算,并把计算结果作为输出信号输出。例如Abs模块(将输入信号去绝对值)、Comples to Magnitude - Angle模块(求一个复数信号的相位和模)等。

(7)Functions & Tables库,包含了描述通用函数和查询表的模块。例如S - Function模块(把S函数文件和Simulink模型结合的模块,它的使用请看后面的章节)、MATLAB Fcn模块(将一个MATLAB函数或者是表达式作用于输入信号)。

(8)Signal & Systems库,它里面的模块允许复合信号(Mux)或者分离信号(Demux),实现外部的输入/输出(Inport和Outport),传递数据到模型中的其他部分(From等),建立子系统(Subsystem,关于子系统我们在后面再详细介绍),以及其他的功能。

而Simulink extras库则包含了一些额外的模块,用户自己建立的模块可以放在这个库里。Simulink还在Blocksets and Toolboxes库提供了一些专用的模块,如用于通信的模块,神经网络应用的模块、控制应用的模块等。读者可以通过在MATLAB命令窗口中输入Simulink3命令来打开模块库,可以把上面的结构看得更清楚。在浏览器里,Blocksets and Toolboxes节点不存在,而是把里面的各个子库直接作为节点。

下而来详细介绍各个子库中的模块的功能。

表4-5至表4-12列出了各个子库中模块的简单说明。

表4-5 Sources库

模块名	功能
Band - Limited White Noise	把一个白噪声引入到连续系统中
Chirp Signal	产生频率增加的正弦信号
Clock	显示或者提供仿真时标
Constant	产生一个常数值
Digital Clock	按指定的间隔产生采样时标
Digital Pulse Generator	产生具有固定间隔的脉冲
From File	从一个文件读数据
From Work space	从在工作空间定义的矩阵读入数据
Pulse Generator	产生固定间隔的脉冲
Ramp	产生一个以常数斜率增加或者减小的信号
Random Number	产生正态分布的随机函数
Repeating Sequence	产生一个可以重复的任意信号
Signal Generator	产生多种多样的信号
Sine Wave	产生正弦波
Step	产生一个单步函数
Uniform Random Number	产生均匀分布的随机数

表 4－6 Sinks 库

模块名	功能
Display	显示其输入信号的值
Scope	显示在仿真过程中产生的信号的波形
Stop Simulation	当它的输入信号非零时，就结束仿真
To File	写数据到文件
To Workspace	把数据写进工作空间里定义的矩阵变量
XY Graph	用一个 MATLAB 图形窗口来显示信号的 X－Y 坐标的图形

表 4－7 Discrete 库

模块名	功能
Discrete Filter	实现 IIR 和 FIR 滤波器
Discrete State Space	实现一个离散状态空间系统
Discrete－Time Integrator	离散时间积分器
Discrete Transfer Fcn	实现一个离散传递函数
Discrete Zero－Pol	实现一个用零极点来说明的离散传递函数
First_order Hold	实现一个一阶保持采样－保持系统
Unit Delay	将信号延时一个单位采样时间
Zero－Order Hold	实现具有一个采样周期的零阶保持

表 4－8 Continuous 库

模块名	功能
Derivative	输出输入信号的微分
Integrator	积分一个信号
Memory	输出来自前一个时间步的模块输入
State－Space	实现线性状态空间系统
Transfer Fcn	实现线性传递系统
Transport Delay	将输入延迟一给定时间
Variable transport Delay	将输入延迟一可变时间
Zero－pole	实现一个用零极点标明的传递函数

表 4－9 Nonlinear 库

模块名	功能
Abs	输出输入信号的绝对值
Algebraic Constraint	将输入信号的约束为零
Combinatorial Logic	实现一个真值表
Complex to Magnitude－angle	输出一个复数输入信号的相角和模长
Complex to Real－Image	输出一个复数输入信号的实部和虚部
Derivative	输出输入信号的时间微分
Dot Product	进行点积
Gain	将模块的输入信号乘上一个增益

(续)

模块名	功　能
Logical Operator	在输入信号实施一个逻辑操作
Magnitude - Angle to Complex	从模长和角度的输入输出一个增益
Math Function	实现一个数学函数
Matrix Gain	将输入乘上一个矩阵
MinMax	输出输入信号的最小和最大值
Product	输出模块输入的乘积或者是 2 商
Real - Image to Complex	将输入信号作为是实部和虚部来合成复数信号的输出
Relational Operator	在输入上进行指定的关系运算
Rounding Function	实现一个舍入函数
Sign	显示输入信号的符号
Slider Gain	按一条斜线来改变标量增点
Sum	产生输入信号的和
Trigonometric Function	实现一个三角函数

表 4-10　Math 库

模块名	功　能
Fcn	将一个指定的表达式到输入信号
Look - Up Table	实现输入的线性峰值匹配
Look - Up Table(2 - D)	实现两个信号的线性峰值匹配
MATLAB Fcn	应用一个 MATLAB 函数或者表达式到输入
S - Function	访问 S 函数

表 4-11　Function & Tables 库

模块名	功　能
Backlash	对一个具有演示特性的系统进行建模
Coulomb & Viscous Friction	刻画在零点的不连续性
Dead zone	提供一个零输出的区域
Manual Switch	在两个信号间切换
Quantizer	按指定的间隔离散化输入信号
Rate Limiter	限制信号的改变速率
Relay	在两个常数间切换输出
Saturation	限制信号的持续时间
Switch	在两个信号间切换

表 4-12 Signal & Systems 库

模块名	功　能
Bus Selector	有选择的输出输入信号
Configurable Subsystem	代表任何一个从指定的库中选择的模块
Data Store Memory	定义一个共享的数据存储空间
Data Store Read	从共享数据存储空间读数据
Data Store Write	写数据到共享数据存储空间
Data Type Conversion	将一个信号转换到另外一个数据类型
Demux	将一个向量信号分解输出
Enable	增加一个使能端到子系统中
From	从一个 Goto 模块接收输入信号
Goto	传递模块输入到 From 模块
Goto Tag Visibility	定义一个 Goto 模块标记的可视域
Ground	将一个未连接的输入段接地
Hit Crossing	检测过零点
IC	设置一个信号的初始值
Inport	为一个子系统建立一个输入端口或者建立一个外部输入端口
Merge	将几个输入线和并为一个标量线
Model Info	显示、修订控制模型信息
Mux	将几个输入信号联合为一个向量信号
Outport	为子系统建立一个输出端口，或者是建立一个外部输出端口
Probe	输出输入信号的宽度、采样时间并且/或者信号的类型
Subsystem	表示在另一个系统之内的子系统
Terminator	结束一个未连接的输出端口
Trigger	增加一个触发端口到子系统
Width	输出输入向量的宽度

第5章　Simulink 详解

5.1　Simulink 的模块和模块库

5.1.1　Simulink 里的模块

模块是 Simulink 建模的基本元素，用 Simulink 建立仿真系统就是选用合适的模块，并用合适的方法连接模块。所以了解各个模块本身的作用，是熟练掌握 Simulink 的一个基础，读者可以根据上一章的提示自己进行摸索，也可以在后面的章节中找到部分模块的参考说明。在 Simulink 里，模块的基本信息可以通过模块数据提示来显示，Simulink 还允许读者自定义数据提示信息的显示内容，对应的方法在上一章中已经有了介绍。这里将涉及的是一些基本概念，这些概念有助于加深读者对模块的理解。

1. 虚模块和非虚拟模块

Simulink 里的模块可以分成最基本的两类：非虚拟模块和虚模块。非虚拟模块在一个系统的仿真中扮演一个重要的角色，向模块模型里增加或者删除一个非虚拟模块，就会改变仿真的行为。而虚模块在系统里则扮演一个次要的角色，虚模块的作用仅仅是有助于用户图像化地组织好模型，而对模块的行为没有任何的贡献。但是这种划分也不是绝对的。有些模块在某些场合是虚模块，而在另一些场合却是非虚拟模块，这种模块在 Simulink 里被称为条件虚模块。

表 5-1 列出了 Simulink 里常见的虚模块以及条件虚模块。

表 5-1　常见的虚模块以及条件虚模块

模块名称	成为虚模块的条件
Bus selector	总是虚模块
DataStoreMemory	总是虚模块
Demux	总是虚模块
EnablePort	总是虚模块
From	总是虚模块
Goto	总是虚模块
Goto Tag Visibility	总是虚模块
Ground	总是虚模块
Inport	总是虚模块，除非它存在于条件子系统内，并且与一个 outport 模块直接相连
Mux	总是虚模块

（续）

模块名称	成为虚模块的条件
Outport	当它存在于任何子系统内(条件或者非条件)它都是虚模块，但是处在模型的最顶层时(也即模型系统里)，就不是虚模块了
Selector	总是虚模块
Subsystem	如果该模块不是条件执行的，它就是虚模块
Terminator	总是虚模块
TestPoint	总是虚模块
TriggerPort	当输入端口不存在时就是一个虚模块

还是以第二章建立的简单模型的示例来说明两者的区别。它的模型图如图 5－1 所示。

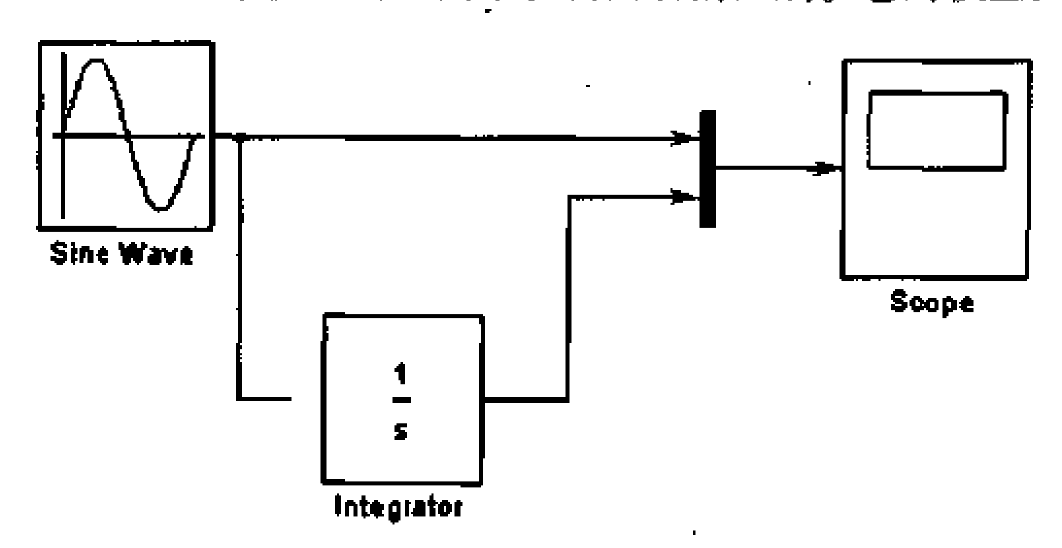


图 5－1　模型图

在图中黑框代表的模块就是 Mux 模块，它就是一个虚模块。Mux 模块的作用就是将输入的多条信号线合为一个向量信号(向量信号的概念见后)，在实际的物理模型里，就像是把几根独立的信号线用绳子扎起来，但是并不会改变信号线所传递的信号。这个模型，其实和下面的模型(图 5－2)等价。

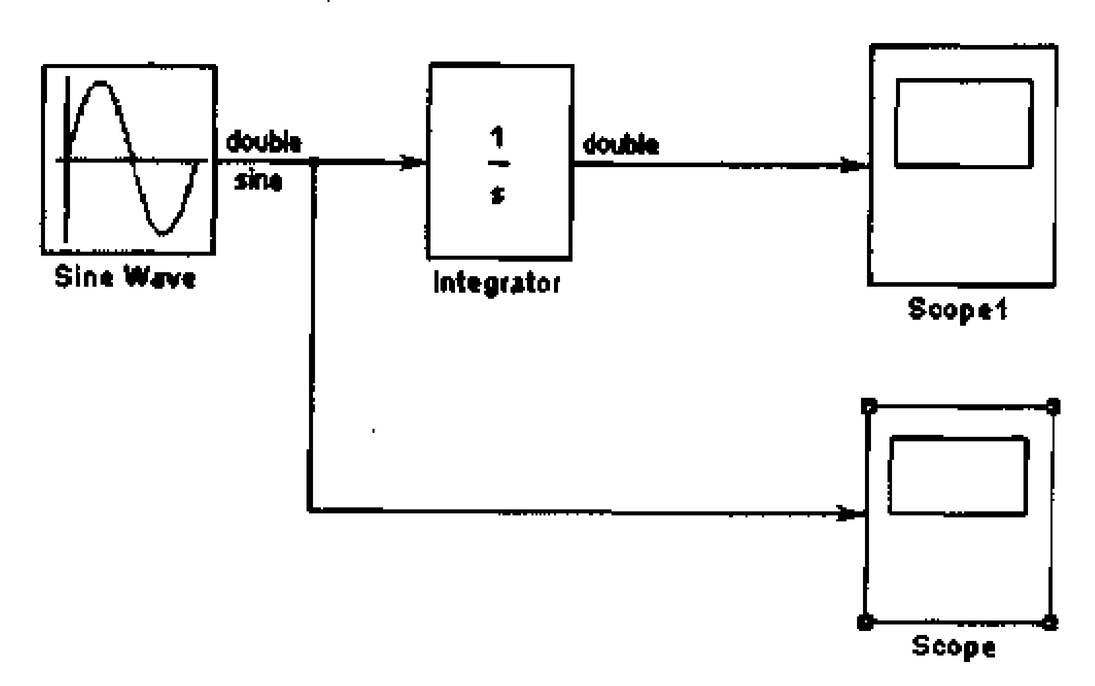


图 5－2　等价图

Mux 模型在图形组织上的作用就是减少了连线的数目和 Scope 模块的数目，只用一个 Scope 模块在一个窗口对照地显示波形，这样使模型显得很有条理。

2. 标量信号和向量信号

向量信号和标量信号的概念在上一章曾粗略地提及，从字面上不难理解，向量信号是多个信号的集合，它对应着实际关系中几条并行连线的合成。在 Simulink 里，充当信号传递的直线只是一个图像化的虚拟形式，实际上信号的传递是依赖变量的，那么向量信号实际上就是指它的输出(输入)变量是一个向量。对于输入信号就不难理解标量信号了。

缺省情况下，大多数模块输出的都是标量信号。对于输入信号，模块都具有一种"智能"的识别功能，能自动进行匹配，这是 MATLAB 语言的一贯风格，实现起来十分简单。读者在学习完本书有关 S 函数的章节后，就能切身体会到这一点。

那么，如何让 Simulink 的模块输出向量信号呢？很简单，只需要设置一个模块参数。还是以前一章建立的那个模型为例，下面的操作将会让 Sine Wave 产生一个向量信号。

请双击 Simulink 用户界面里的 Sine Wave 图标，打开它的参数设置对话框，按图 5-3 所示设置参数。

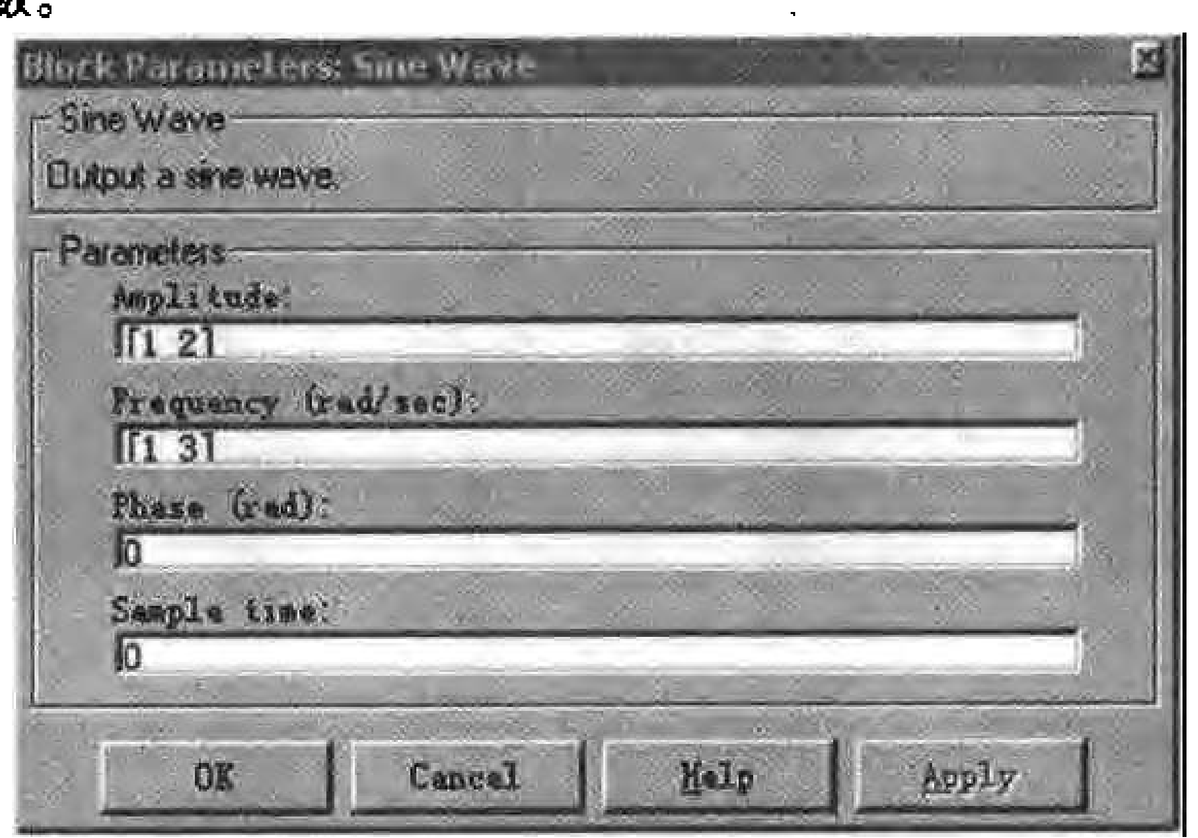


图 5-3 设置 Sine Wave 的参数

点击 OK 按钮关闭参数设置对话框后，让模型图表显示向量信号的宽度，就可以看出 Sine Wave 的输出端是一个两位的向量信号。图 5-4 描绘了它的样子。

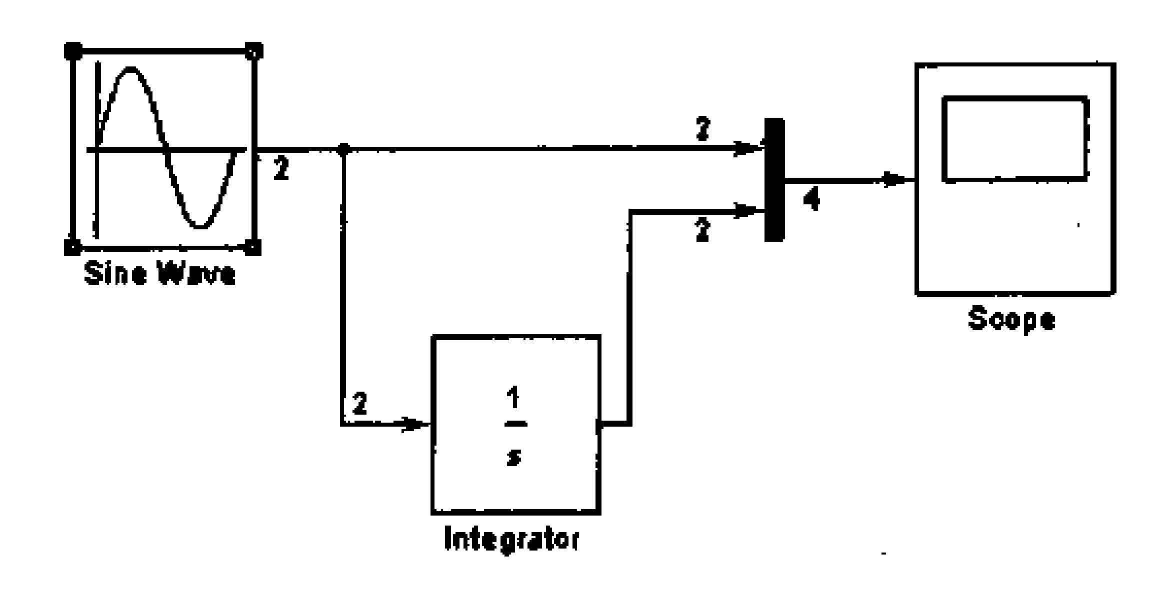


图 5-4 Sine Wave 输出了两位的向量信号

图 5－5 是参数设置改变后获得的仿真结果。从图中可以看出，Sine Wave 输出了一个两位的向量信号，即包含了两个幅度和频率均不同的正弦信号。这个向量信号和它经过积分后获得的向量信号被虚模块 Mux 合并为一个 4 位的向量信号，这个向量信号在 Scope 里的显示波形就是图 5－5 所示的。

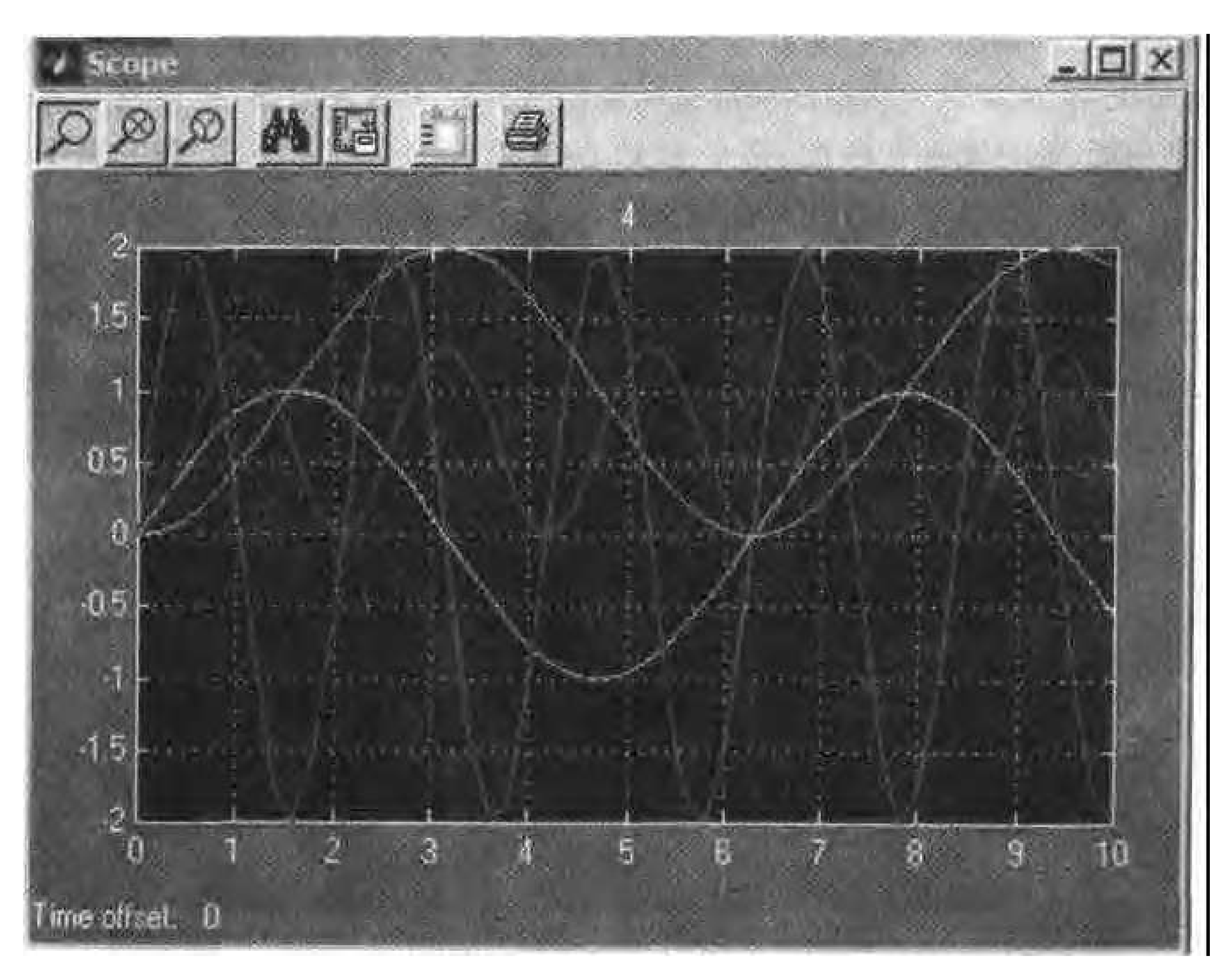


图 5－5 参数改变后的仿真输入图

这个例子说明了，Sine Wave 模块输出的信号宽度受参数的控制。当参数设置为一个向量时，例如上面的[12]，它表明该模块输出的第一个正弦信号幅度为 1，第二个信号幅度为 2，也就是输出了一个两位的向量信号。但是要注意，模块中所有的参数都是按向量元素的位置来和信号对应的，不存在自由组合的问题。例如上例中的幅度和频率分别设置成[1 2]和[1 3]，这样不是说产生四个信号，而是两个信号。它们的参数如果用[幅度 频率]来表示，则分别是[1 1]和[2 3]。此外，在设置向量参数时，一定要注意不同参数之间向量大小的匹配问题。Simulink 有时并不要求所有的参数都设置成同样大小的向量，例如，幅度[1 2]，频率[1]是可以的。但是[1 2 1]和[1 2]的搭配却会出现错误提示，读者可以自己体会。

以上说的只对像 Sine Wave 那样的数据源模块才适用，它们的输出完全视设置的参数而定。但 Simulink 里的更多模块是既有输入端又有输出端的，一般它们的输出信号受输入信号和参数共同控制，例如 demux 虚模块。

Simulink 有很多模块具有多个输入端，如果使用时，读者把标量信号和向量信号混合输入，例如 Sum 模块的两个输入端，一个是标量信号，另一个是向量信号，这时 Simulink 会怎样处理呢？下面建立一个简单的模型做一下试验。

图 5－6 是该模型建好后的情况。表 5－2 列出了模型所需的模块以及各自的参数设置。

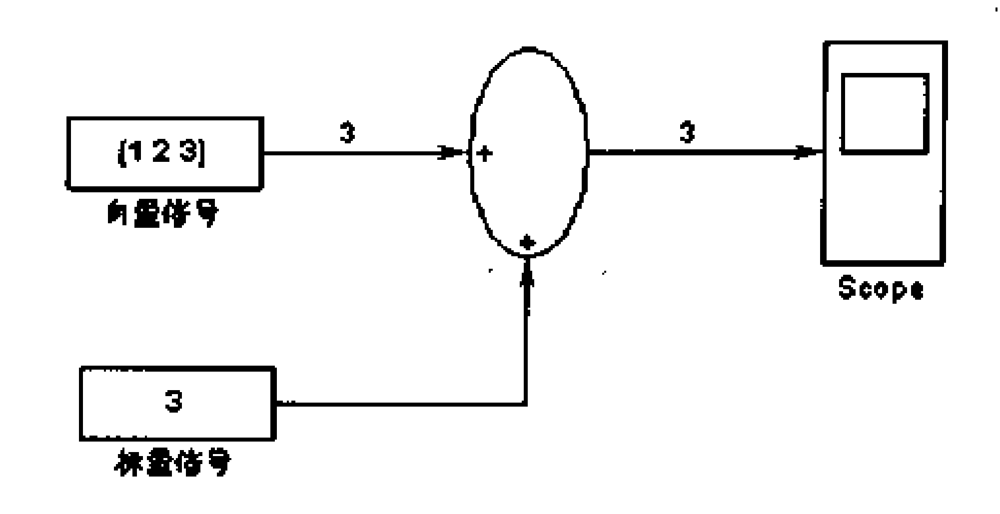


图 5-6　混合输入试验模型

表 5-2　所需模块及参数设置

图标标签	模块类型	所在子库	参数设置
向量信号	constant	Source	Constant value [1　2　3]
标量信号	Constant	Source	Constant value 3
	Sum	Math	缺省设置
scope	Scope	Sink	缺省设置

运行仿真，会发现 Scope 的显示区域出现了三条直线，其幅度分别为 4、5、6，也就是说 sum 的输出是一个向量信号。

这个现象在 Simulink 里称为输入信号的标量扩展，即把输入标量扩展成和向量信号同样大小的向量，新扩展的向量信号的元素就是原来的标量信号。注意，这种处理不能推广到两个不同长度的向量的混合，因为这时候 Simulink 无法决定扩展的方法。例如一个是两位的向量信号，一个是三位的向量信号，Simulink 对两位的向量信号扩展时无法决定是用第一位还是用第二位。这个道理也可以解释设置模块参数为向量时遇到的匹配问题，其实，也可以把设置参数为向量理解成一种扩展。

在 Simulink 里，还有一种称为参数的标量扩展，如图 5-7 所示的例子。

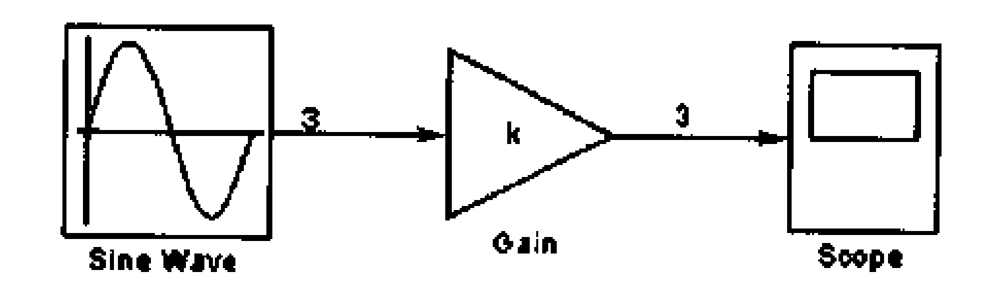


图 5-7　参数的标量扩展

在图 5-7 中，Sine Wave 的幅度属性设成[1 2 3]，而 Gain 属性也设置成[1 2 3]，它的意思是把输入向量信号的三个信号按顺序分别放大 1、2、3 倍，运行模型时可以看到这个结果。当 Gain 属性设置成一个标量，例如 1 时，这时 Simulink 自然会将这个参数扩展成和输入同长度的向量，这里是[1　1　1]。这就是所谓的参数标量扩展，它其实是很简单的概念。

5.1.2 Simulink 的模块库

库技术使得用户能从外部库里拷贝到自己需要的模型，并且在库里的模型变动时能够自动地更新用户先前已拷贝在模型里的模块，而不用用户手动更改。为了更好地理解 Simulink 库的这种功能，我们先介绍几个术语。

库——模块的集合。

库模块——库里的模块。

引用模块——库模块的拷贝。

链接——引用模块和库模块之间的连接，它使得 Simulink 在库模块发生变换时，能够更新引用模块。

拷贝——从库模块或者别的引用模块产生一个引用模块的操作。

图 5－8 反映了这些术语间的关系。当用户从库里用拖动操作将库模块拷贝到用户的模型里时，Simulink 会在模型里建立一个参考模块。用户可以设置参考模块的参数，但是不能封装(关于封装，在 5.5 节说明)参考模块，对于封装好的模块，用户也不能修改封装。注意，Simulink 不是简单地复制出一个参考模块，它同时还建立一个从参考模块到它的对应源库模块的链接。这个链接的作用是在参考模块里保存了它的源模块的信息，以便在用户更新图表时，Simulink 能按照模型中引用模块的保存的信息，搜寻它的对应库模块，并根据它重新产生一个参考模块来代替原模块。

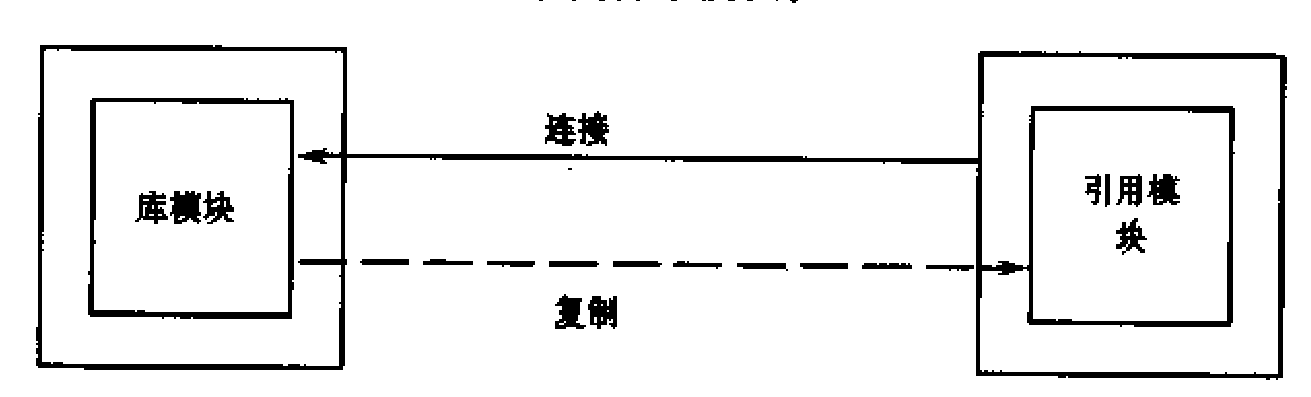


图 5－8 库模块和引用模块之间的关系

确切地说，参考模块和库模块是靠名称链接起来的，也就是参考模块所链接的特定模块在进行复制时名称才有效，这个意思是很明白地。

当 Simulink 试图更新参考模块时，如果它在 MATLAB 路径上找不到库模块和源库，那么这个链接将变成无法确定了。Simulink 产生一个错误信息，并将这些参考模块用红色的虚线表示。图 5－9 显示了链接无法确定的情况。

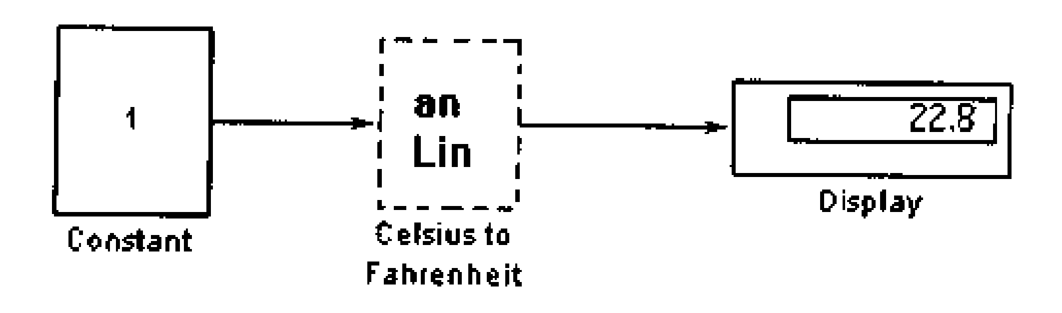


图 5－9 链接不确定的情况

读者可以按照下面的步骤来获得这个效果。首先,新建立一个模型,分别从 Source、Simulink extras 和 Sink 库里取出相应的模块,按图 5-9 把线连接好,最后别忘了保存模型;然后按照图 5-10 所示进行操作。

将 Transformations 图标拖到你的窗口中,然后用左键双击此图标,打开 Transformations 库,如图 5-11 所示。

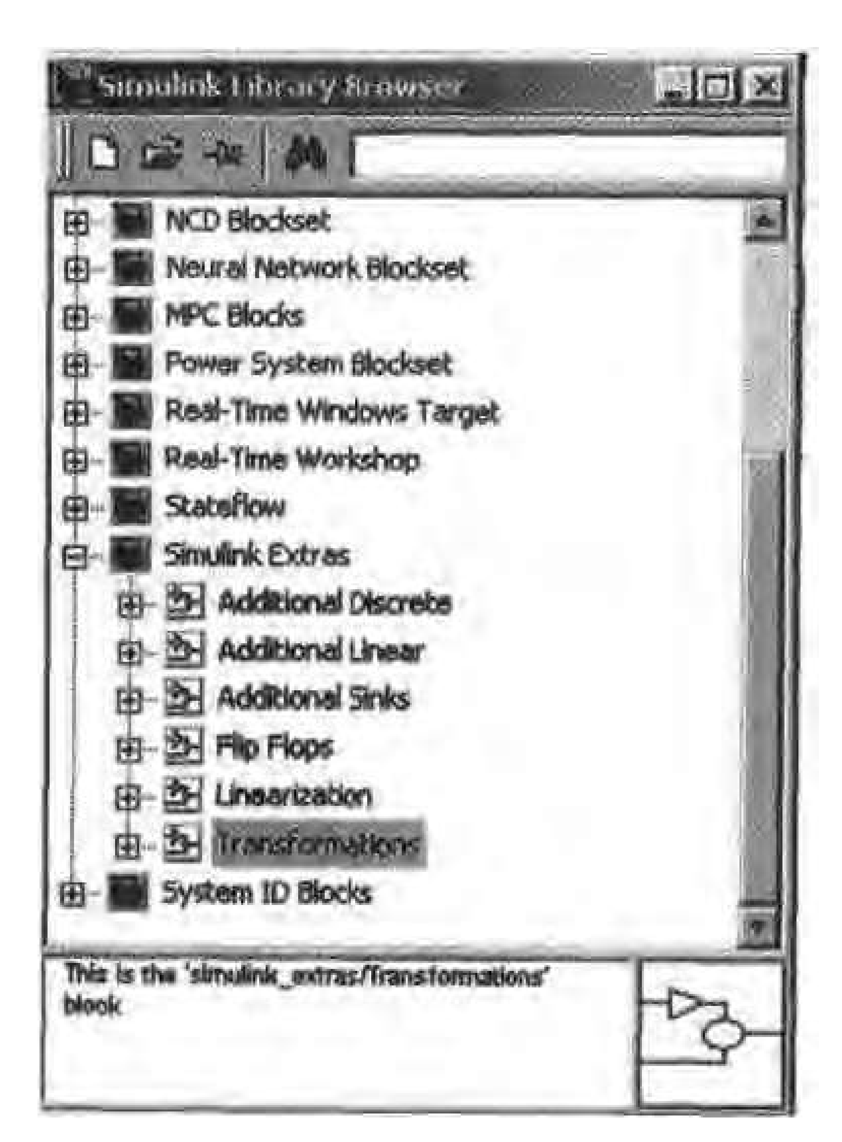



图 5-10 打开 transformations 库

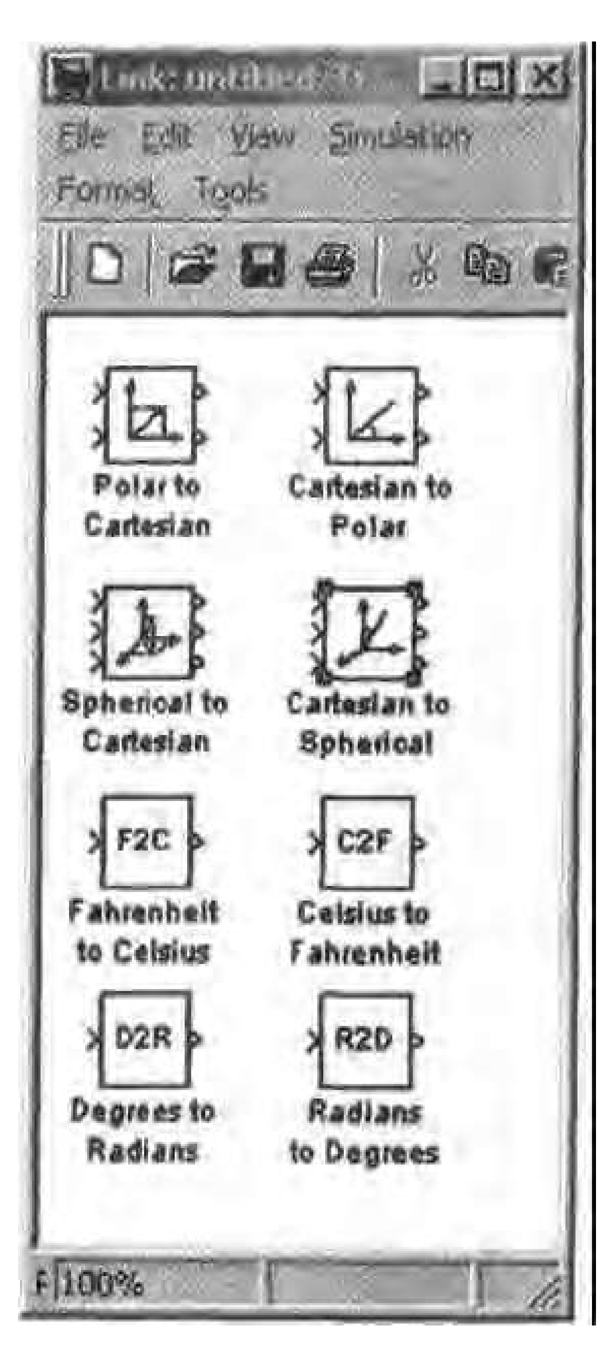


图 5-11 transformations 子库

读者请先将库另存为其他名称的库作为备份,这个操作将另存整个 Simulink _ extras 库,而不仅仅是 Transformations 子库。读者在另存为时,最好是不要改变目录,这样万一出现错误时,就可以把库复原。复原的方法很简单,用备份库覆盖已经损坏的 Simulink _ extras 库,并把它的文件名改为 Simulink _ extras;然后用 Edit 菜单下的 Unlock Library 命令,使库成为可以编辑状态,接着把 Selsius to Fahrenheit 模块删除,再把库保存。完成这些操作后,再回到原来的模型窗口,单击 Edit 菜单下的 updata datagram 项,就会出现图 5-9 所示的内容。

可以按下面的几种方法来修复无法确定的链接。

(1)删除未链接的参考模块,将库模块重新复制到模型里。

(2)把包含所要求模块的目录加到 MATLAB 的搜索路径里,然后再从模型窗口选择 Updata Datagram 命令更新图表。

(3)双击参考模块,在弹出的设置对话框里,改正路径,按 Apply 按钮关闭对话框。对于前面所举的例子,由于子库模块已经被删除,要修复无法确定的链接只能重新建立一个库

模块,然后将它重新和参考模块对应起来。但是这里,只需要用备份库复原原来的库即可。

在 Simulink 里,用户可以通过获得模型的库信息来查询模块的链接状态。库信息包括下面几个部分。

- 模块——模块路径;
- 库——库的名称;
- 引用模块——所引用的模块路径;
- 链接状态——表示链接的状态,确定或无法确定。

要获得该信息,可以用下面的 MATLAB 命令。

```
>>libdata = libinfo ('thermo')
libdata =
3x1 struct array with fields:
        Block
        Library
        ReferenceBlock
        LinkStatus
```

libdata 是一个结构。可以用这样的格式访问结构的字段:结构名.字段名。例如要查看 Block 字段值可以输入

```
>>libdata.Block
    ans =
    thermo/Celsius to
    Fahrenheit
    ans =
    Thermo/Fahrenheit
    to Celsius
    ans =
    thermo/Fahrenheit
    to Celsius
```

这个结果说明,在 thermo(前面提到的演示模型)里有三个模块是参考模块,Block 字段显示了参考模块的名称。

注意,MATLAB 里的命令和变量名都是大小写敏感的,所以在输入上面的变量时一定要注意大小写,比较好的办法是直接拷贝过来。

下面的注意力将集中到链接状态上。

链接状态可以取下面几个值:

- 'none',表示该模块不是一个参考模块;
- 'resolved',表示该模块是参考模块并且链接是确定的;
- 'unresolved',表示该模块是一个参考模块,并且链接是无法确定的。

可以通过 Edit 菜单下的 Break Library Link 来去掉参考模块的链接,这样库模块的任何变动将不会影响用户模型中的模块(此时,已经不是参考模块了)。

或者用下面的 MATLAB 命令来实现：

≫set _ param ('linkexample/')

这个命令的意思是把链接状态设置成'none'，也就是使模块不再是参考模块。注意这种过程是不可逆的，你不能把一个不是参考模块的模块变成为参考模块，因为模块原来保存的信息消失了，Simulink 不知道链接到哪个模块。

同 MATLAB 语言一样，Simulink 也提供了扩展系统的能力。它允许用户生成自己的库，用户可以把一些常用的模块放在一个库里，免得寻找时的麻烦。在后面的章节里将会教会读者创建自己的模块以及库。

选择 Simulink 用户界面 File 菜单下的 New Library 命令，弹出了一个空白的库窗口。然后可以把读者常用的模块复制到该窗口。因为你复制到新建库里的模块也仅仅是参考模块，为此选中所有的模块(CTRL + A)，用 Edit 菜单下的 Break Library Link 去掉所有参考模块的链接，这样它们就可以作为库模块使用了，最后保存所建的库。图 5 - 12 是建好的库的示意图。

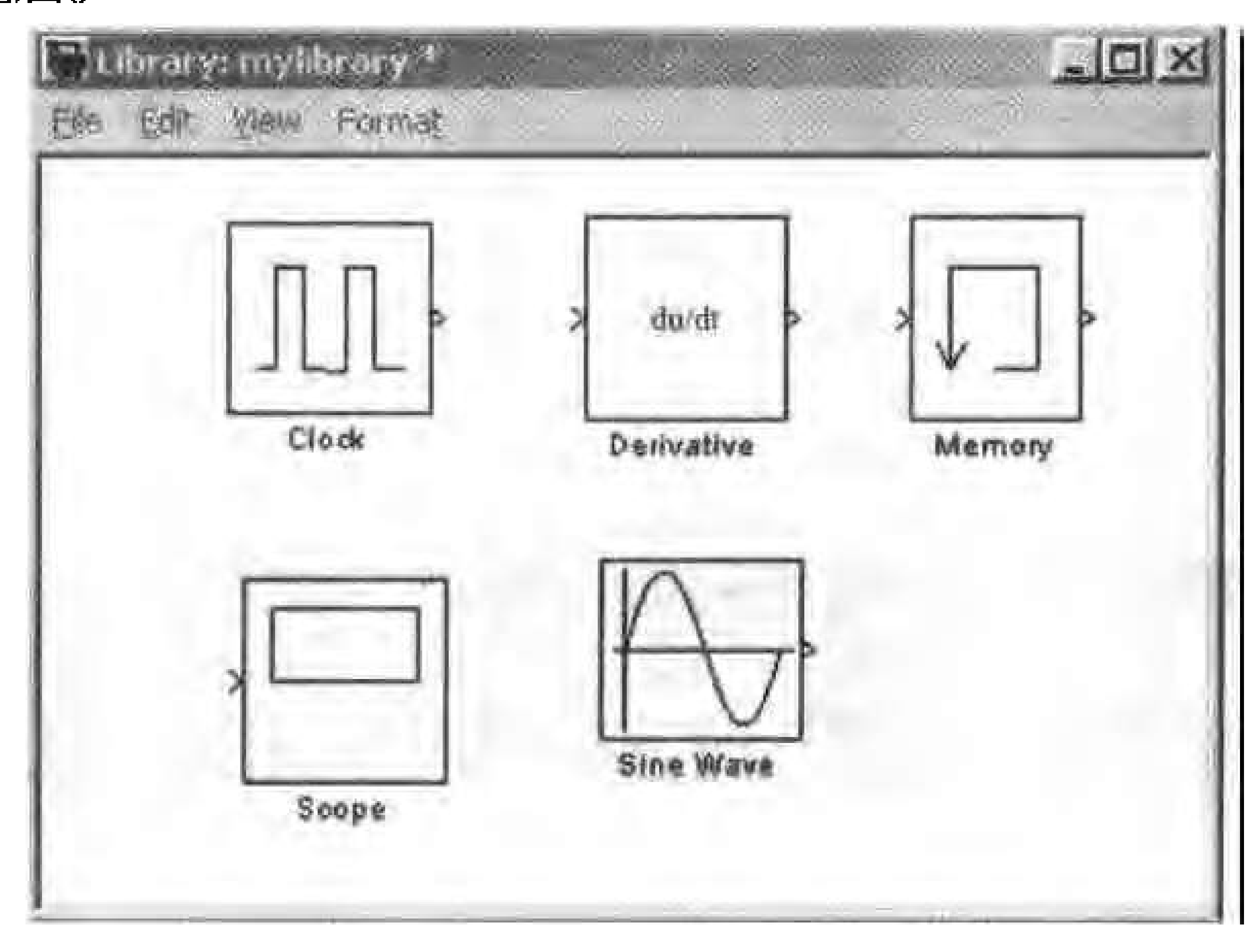


图 5 - 12 自定义的库

但是，这个库有一个缺点：它不能用库浏览器，在下一节将会介绍如何来定义能用浏览器浏览库的方法。

以上讲述的命令操作在 Simulink4.0 里，与参考模块和库模块间的链接相关的操作有一些细微的变化。具体体现在 Edit 菜单，用 Link Options 子菜单替代了 Break Library Link 命令。而 Break Library Link 的功能用 Link Options 子菜单里的 Disable Link 命令来完成，Link Options 里还提供一个 Restore Link 命令用于恢复被禁止的链接。

Simulink4.0 新增的另一功能是传递链接修改。前面说过具有激活的链接的模块不能具有结构性的变化。当用户试图恢复一个具有结构性变化的链接时，Simulink 会提示用户是传递还是放弃这个变化。如果用户选择传递，Simulink 会根据修改的参考模块来更新库模块，也就是提供了一个参考模块的反作用。如果用户选择放弃变化，Simulink 会

用库模块替换已被修改的参考模块。当用户要恢复一个具有非结构性变化的链接，Simulink 不用提示是传递还是放弃变化，而直接将链接激活。

用户也可以通过 Link Options 子菜单里的 Progagate/Discard Changes 命令来手动确定对一个参考模块所作结构性变化是传递还是放弃。使用这个命令，当然也要先选择一个参考模块。要想看参考模块和库模块之间的非结构性参数变化，可以选择 Link Options 子菜单的 View 命令。

5.2 模拟方程

如何模拟方程是 Simulink 初学者最困难的问题之一，在这一节里将给出一些这方面的例子，通过例子来使读者加深理解。

1. 将摄氏温度转换为华氏温度

第一个例子是模拟把摄氏温度转换为华氏温度的方程：

$$T_F=(9/5)\,T_C+32$$

首先，来考虑一下所需要的模块：

(1)Ramp 模块，用来产生温度信号，所处库为 Source 库；

(2)Constant 模块，用来产生一个常数 32，同样来自 Source 库；

(3)Gain 模块，将输入信号乘上 9/5，来自 Math 库；

(4)Sum 模块，把两个量加起来，也来自 Math 库；

(5)Scope 模块，显示输出的结果，来自 Sink 库。

然后，把所有的模块复制到模型窗口中，如图 5－13 所示。

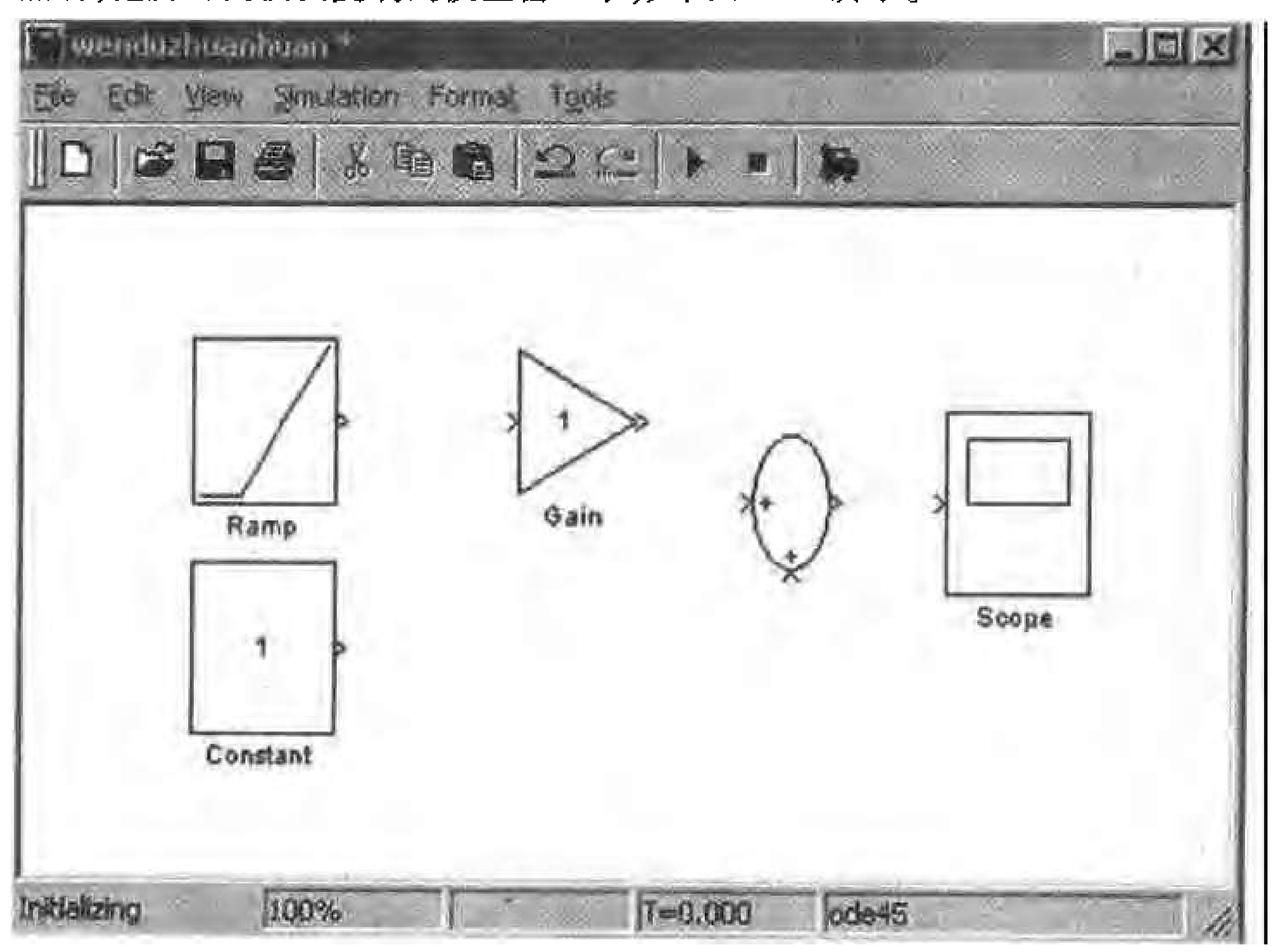


图 5－13 将所有需要的模块复制到模型窗口

给 Gain 模块和 Constant 模块设置合适的参数值(Gain 为 9/5,Constant 为 32),就是双击模块方框,打开参数设置对话框输入相应的数值。下面,就可以连接各个模块了,请按照图 5-14 所示进行连接。

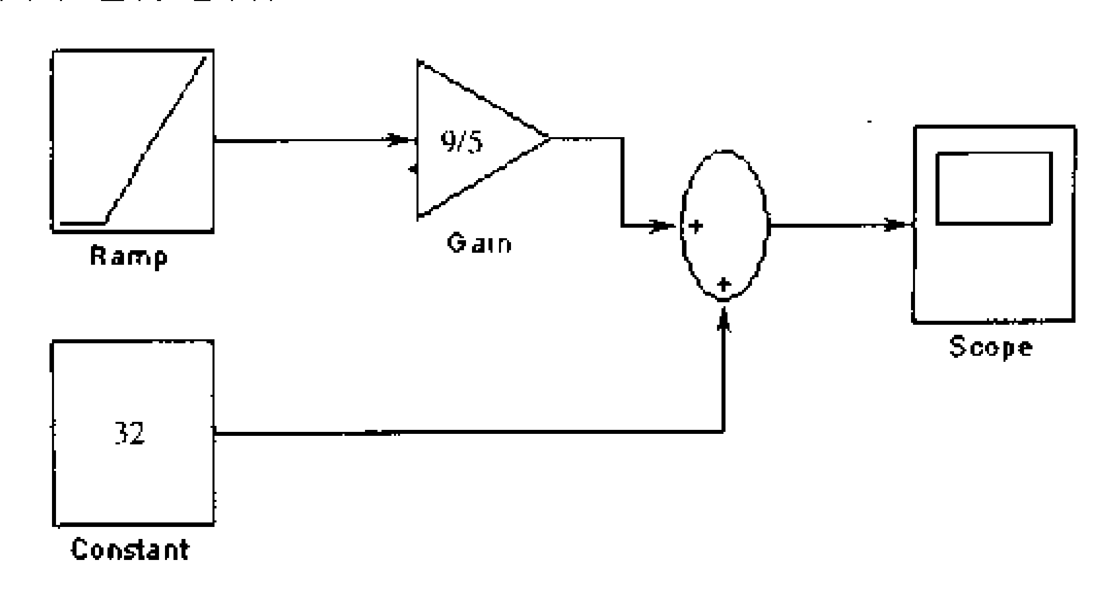


图 5-14　连线好的模型

这样,将摄氏温度转化成华氏温度的模型就建好了,你可以运行仿真,来观看 Scope 模块的输出。如图 5-15,显示了转换前后的情况,左图是摄氏温度,右图是相对应的华氏温度。

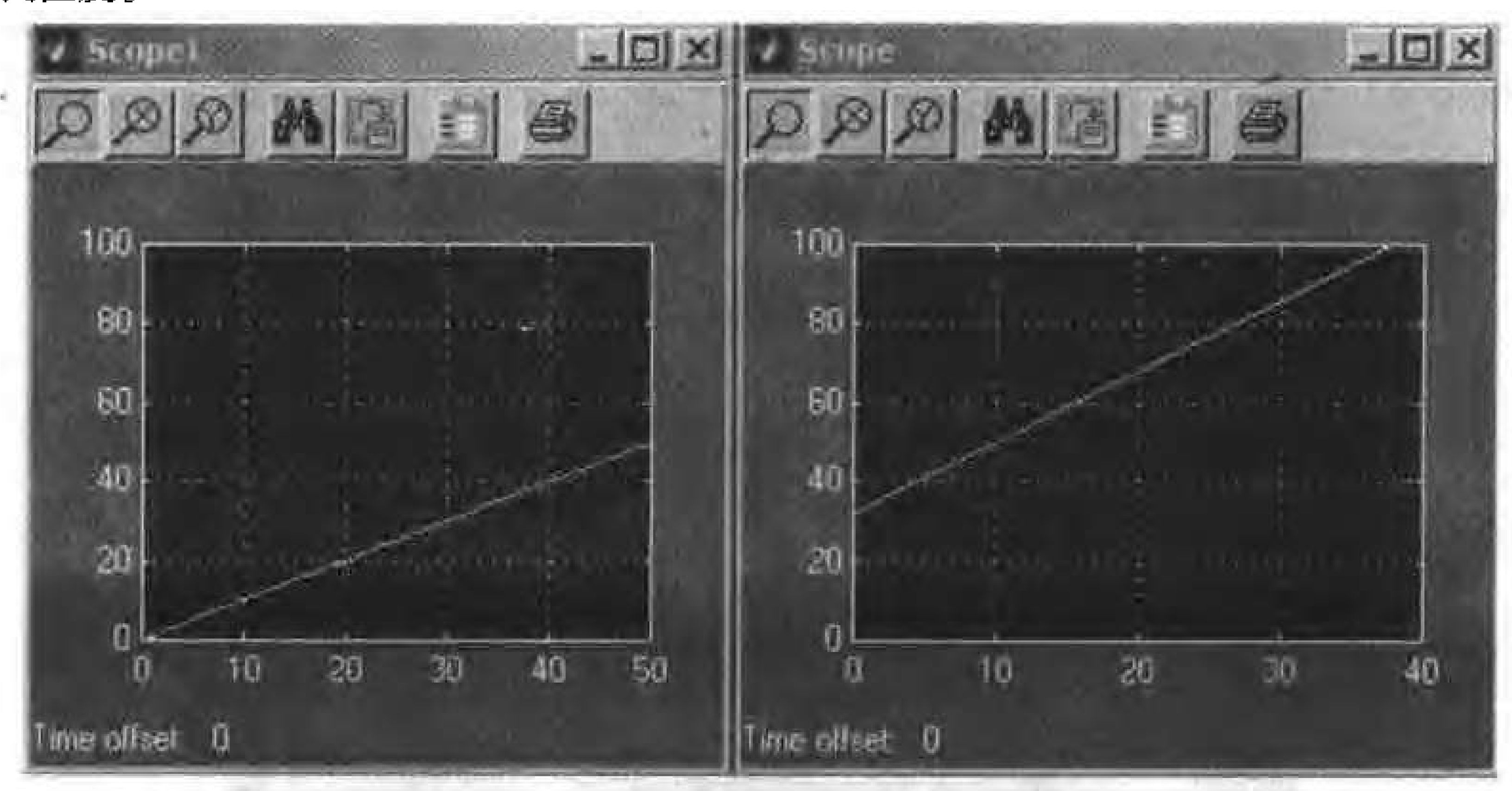


图 5-15　仿真结果图

2. 对简单的连续系统进行建模

下面来演示如何对微分方程进行建模,看如下的方程:

$$\dot{x}(t) = -2x(t) + u(t)$$

其中,$u(t)$是幅度为 1,频率为 1rad/s 的方波信号。积分模块将 $x(t)$的微分信号积分来获得 $x(t)$。此模型中需要的其他模块包括一个 Gain 模块和一个 Sum 模块。要产生方波信号,可以使用 Signal Generator 模块,选择波形为方波(Square)并改变频率单位为 rad/s。同样,用 Scope 模块来观看最后的输出结果。

把所需要的模块复制好后,把 Gain 模块的增益参数设为 -2。然后按图 5-16 所示

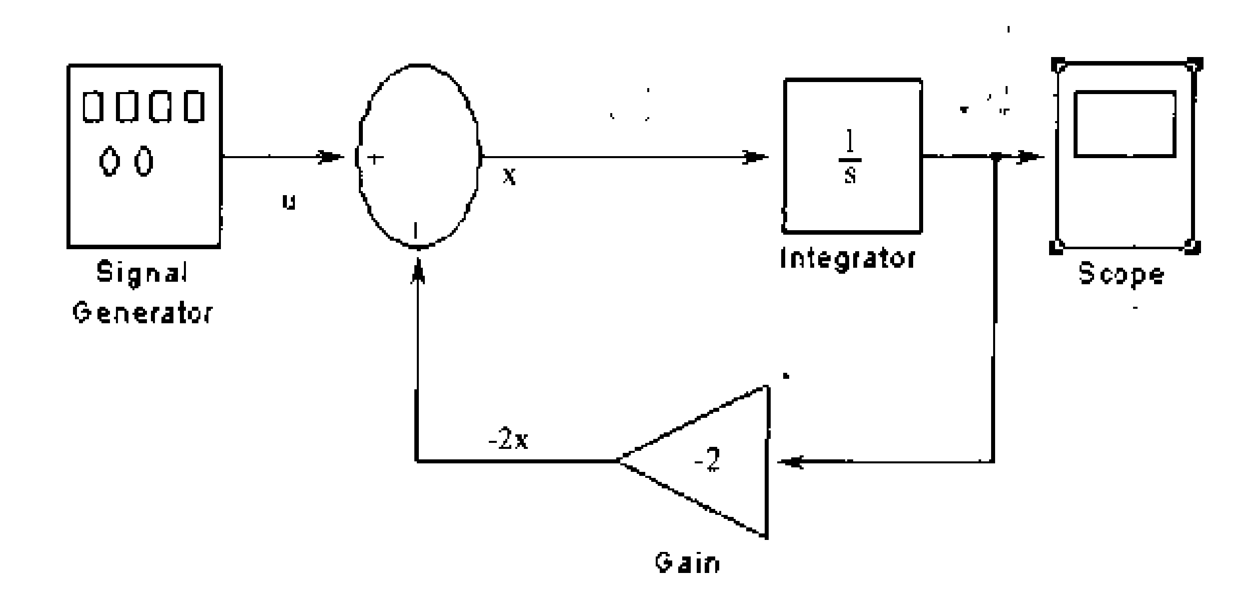


图 5-16 连好线后的模型图

进行连线。

图中，Gain 模块的翻转，可以通过 Format 菜单下的 Flip Block 命令来进行。

这个模型的一个重要特点是包含了一个由 Sum 模块、Integrator 模块和 Gain 模块组成的环路。在这个方程里，x 是 Integrator 模块的输出，它同样是计算 x 的微分的模块的输入，这个关系就是通过模型中的环路来实现的，运行仿真后，Scope 模块显示的波形如图 5-17 所示。

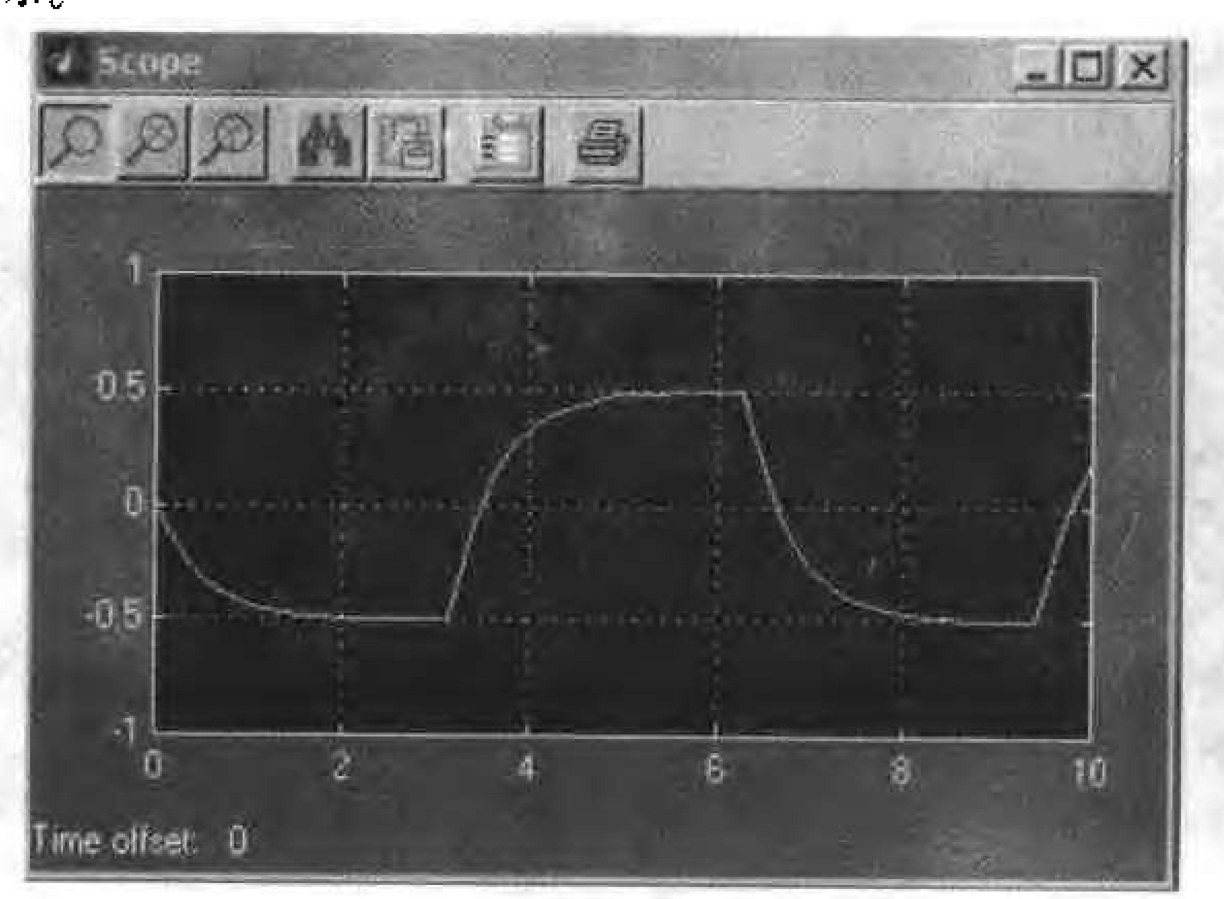


图 5-17 仿真输入波形

本例所处理的同样可以表示为传递函数的形式。很显然，上面的微分方程左右取拉氏变换，即：

$$x'(t) = -2x(t) + u(t)$$

变为

$$sX(s) = -2X(s) + U(s)$$

$$\Rightarrow \frac{X(s)}{U(s)} = \frac{1}{s+2}$$

这样就不难推出由方程描述的输入 $u(t)$，输出 $x(t)$ 的系统的传递函数为

$$Fcn(s)=\frac{1}{s+2}$$

于是，就可以用 Transfer Fcn 模块来进行建模(Continuous 库)。为此，要先指定对应的传递函数，可以在模块参数对话框设置 Nemerator(分子)参数、Denumerator(分母)参数分别为[1]和[2]。也就是说，这个模块是用传递函数的分式形式表示，且系数从左至右按 s 的降幂排列。于是，上面的模型就可以简化为图 5 - 18。

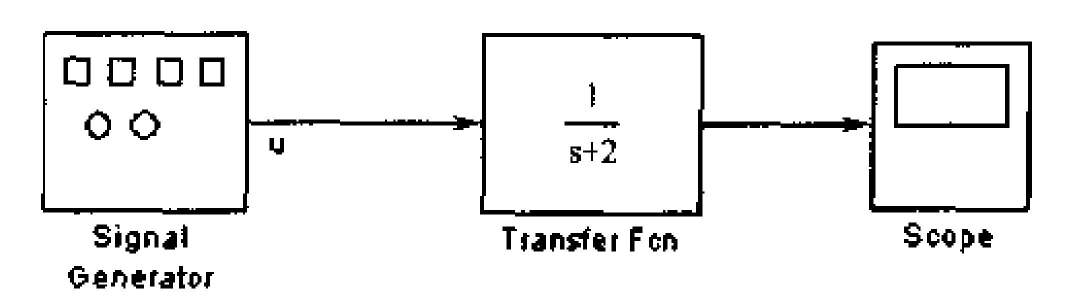


图 5 - 18　简化后的模型

5.3　Simulink 里的数据类型

众所周知，数据类型决定了分配给一个数据的存储资源，决定数据表示的精确性、动态范围、性能和存储资源的利用。和 MATLAB 一样，Simulink 也允许用户说明 Simulink 模型中信号和模块参数的数据类型。

这种能指定模型的信号和模块参数的数据类型的能力，在实时控制应用中非常有用。例如，它允许一个 Simulink 模型指定一个最优的数据类型在由代码生成工具从模型生成的代码里表示信号和模块参数。这种代码生成工具比如在后面章节中将讲到的 Real - Time Workshop，通过选择合适的数据类型，用户可以大大增加代码的性能，而又缩小代码的大小。

Simulink 在开始仿真之前以及仿真过程中，会进行一个额外的检查，以确认运行的模型的类型安全性。所谓模型的类型安全性，指从模型产生的代码不会出现上溢和下溢，并非因此产生不精确的结果。使用 Simulink 缺省的数据类型(double)的模型都是固定有类型安全的。如果读者可以确信自己不会建立非缺省数据类型的模块，那么这一节对读者的意义不大。但是，话又说回来，读者最好还是仔细阅读一下这一节。

5.3.1　Simulink 支持的数据类型

Simulink 支持所有的 MATLABA 内置数据类型，内置数据类型指 MATLAB 自己定义的数据类型，以区别于用户自己定义的数据类型。除了特别说明，本书中的数据类型都是内置数据类型。表 5 - 3 列出了所有的 MATLAB 内置数据类型。

表 5 - 3　MATLAB 内置数据类型

模块名	说　明
double	双精度浮点类型
Single	单精度浮点类型
Int8	有符号 8 位整数

(续)

模 块 名	说 明
Unit8	无符号 8 位整数
Int16	有符号 16 位整数
Unit16	无符号 16 位整数
Int32	有符号 32 位整数
Unit32	无符号 32 位整数

除了内置数据类型，Simulink 还定义了布尔类型，取值位 0 或 1，它们的内部表示式 unit8(无符号 8 位整数)。

所有的 Simulink 模块都缺省地接收 double 类型地的信号，但有些模块需要布尔类型的输入，而另外一些模块支持多数据类型输入，还有解支持复数信号。表 5－4 列出了这些模块。

表 5－4 Simulink 里支持复数的模块

模 块 名	说 明
Abs	输入或者输出具有 double 类型的实数或者复数信号，输出 double 类型的实数
Combinatorial Logic	输入、输出的类型，在布尔模式使能时只能为布尔类型，否则可以为 double
Constant	输出具有任何类型的实数或者复数信号
Data Type Conversion	输入、输出具有任何类型的实数或者复数信号
Demux	接收混合类型的信号向量
Display	接收具有任何复数或者实数数据类型的信号
Dot Product	输入和输出 double 类型的实数或者复数信号
Enable	相应的子系统使能端接收布尔或者复数信号
From	输出连接到对应 Goto 模块的信号的数据类型
From Workspace	输出类型为对应的工作空间的数据值的类型
Gain	输入可以是任何数据类型的实数或者复数信号的向量
Goto	输入可以是任何数据类型
Ground	输入与之相连信号相同类型的零信号
Hit Crossing	输入 double 类型的信号，输出布尔类型
Inport	可以接收任何数据类型的信号，当位系统层的输入端或者是直接和同子系统的输出端直接相连时，那么它的各个元素信号的数据类型必须相同
Integrator	该模块在它的数据端口接收 double 类型，在置位端接收 double 类型和布尔类型
Logical Operator	输入和输出布尔类型的实信号，不在布尔模式还可处理 double 类型
Manual Switch	接收任何类型的实数或者复数信号，所有输入端信号的类型必须相同
Math Function	输入和输出 double 类型的信号

关于模块的输入、输出信号支持的数据类型的详细说明，见本书的附录。一个模块如果没有说明它支持的数据类型，那就表示它只能支持 double 类型的数据。

在设置模块参数时,指定一个值为特殊数据类型表示的方法为:type(value)。例如要把常数模块的参数设为 1.0 单精度表示,则可以在参数对话框输入:

≫single (1.0)

而把一个具有特定数据类型的信号引入模型的方法有:

(1)通过根目录 inport 或者 FromWorkspace 模块将 MATLAB 工作空间中具有指定数据类型的信号数据导入你的模型;

(2)建立一个常数模块并设置它的参数为指定的类型;

(3)使用 DataTypeConversion 模块将一个信号转换为指定的信号。

若要显示一个信号的数据类型,可以从 Format 命令选择 Port Data Types 命令,对于这一命令的用法,在前面已经提到过。

5.3.2 数据类型的传递

无论用户何时开始仿真,显示端口数据类型和更新数据类型显示,Simulink 都会进行一个称为数据类型传播的处理步骤。这个步骤确定没有被特别设定的信号的数据类型,以及检查信号的数据类型和输入端口是否冲突。如果冲突,Simulink 会显示一个提示对话框,告诉用户出现冲突的信号和端口,它还会把冲突信号的路径用亮条加以显示。读者可以在模型里插入一个 Data type conversion 模块来解决类型冲突。

下面的几条法则将有助于读者建立类型安全、运行不出错的模型。

(1)信号的数据类型一般情况下不会影响参数的数据类型。但是有个重要的例外,那就是常数模块,因为它的输出值决定于所设置的参数,所以输出端口的数据类型就决定于所设置的参数。

(2)如果模块输出端的信号是输入信号和模块参数的函数,并且输入信号和参数的数据类型不一致,Simulink 在计算模块输出之前会自动地把参数的数据类型转化成输入类型。

(3)总的来说,输出的数据类型决定于模块的输入数据类型,除了两个例外:Constant 模块和 Data type conversion 模块,它们的输出类型是由参数的类型决定的。

(4)虚模块接收任何数据参数类型的信号作为输入。

(5)连接到非虚拟模块同一端口的向量信号的元素信号必须具有相同的数据类型。

(6)连接到非虚拟模块输入端口的所有信号必须具有相同的类型。注意和第(5)点的区别,这里是指信号之间的关系,而第(5)点强调的是同一元素的数据类型。

(7)控制端口接收布尔类型或者 double 类型的信号。

(8)Solver 模块只接收 double 类型的信号。

(9)禁止过零检测的模块只能和非 double 数据类型的信号相连接。

缺省情况下,当 Simulink 检测到接收 double 类型数据的模块和只选择布尔类型的模块相连时,不会给出错误提示,这主要是为了和以前的版本的模块相兼容。用户可以通过在 Simulink Parameters 对话框的 Diagnostics 页,去除 Relax boolean type checking 选项的选中状态,来使 Simulink 实施严格的布尔类型检验。

前面曾讲过,在仿真过程中,如果输出信号是模块输入和参数的函数,计算这种输出信号时,Simulink 会把参数类型转换到信号的数据类型。对于这条法则,有下面的几个特

例：

(1) 如果信号数据类型无法表示参数值，Simulink 将中断仿真，并给出错误信号。

请看下面的模型(图 5－19)，表 5－5 列出了模型中各个模块的参数值。

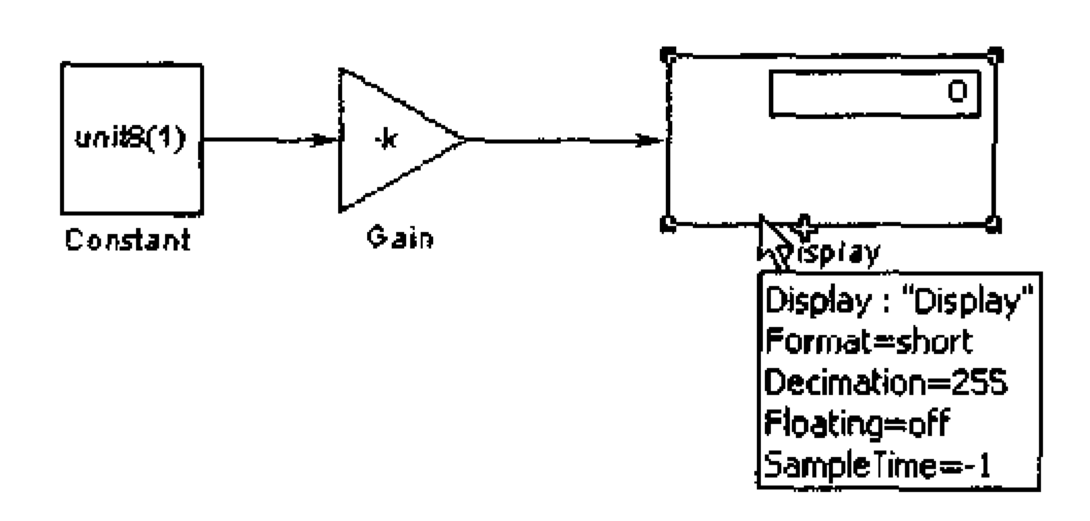


图 5－19　示例模型

表 5－5　模型所需要的模块

模块名称	参数设置
Constant	Consant: unit8(1)
Gain	Gain: int32 (255)

这个模型用 Gain 模块将 Constant 模块的输入信号放大，计算 Gain 模块时需要计算输入信号和增益的乘积。这就是要求两个值的类型相同。然而，这个模型中 Constant 模块的值是无符号类型，而 Gain 模块的值却是 int32 类型的。于是，Simulink 就要进行一个类型转换，把参数的类型映射到信号类型。如果信号的数据类型能够表示参数的数据类型，Simulink 就完成这种转换。因为 Gain 的值是 255，恰巧在无符号数的表示范围(0～255)内，所以 Simulink 可以完成这种转换，仿真运行不会产生错误。

但是如果把 Gain 模块的参数 Gain 改成 256，那么就超出了无符号数的表示范围了，Simulink 无法进行转换，它会给出一个错误的信号。

(2)如果信号的数据类型能够表示参数的值，仅仅是表示损失的精度，Simulink 会继续仿真，并在 MATLAB 命令窗口给出一个警告信息。例如把上例中的 Gain 参数设为是 double(25.4)，由于它处在无符号数表示范围内(0～255)，Simulink 就会把 25.4 的整数部分截取，并转换成无符号数，所以最后的结果是 25，但是给出了一个精度损失警告。如果是 25.0 就不存在精度损失，所以也不会有警告信息。

5.3.3　在模型里使用复数信号

缺省情况下，Simulink 的信号的值都是实数，然而，模型也可以建立和处理值为复数的信号。

有如下几种方法可以把复数信号引入到模型中：

(1) 把 MATLAB 工作空间里的复数信号通过 root－level import 导入到模型里；

(2) 复制一个 Constant 模块到你的模型里，并把它的参数设为复数；

(3) 分别产生对应着复数信号实部和虚部的两个实信号，然后把它们联合成一个复数信号。这里要用到 Real - Image to Complex 模块，它的位置在 Simulink 库的 Math 子库，在 Math 子库里还有一个 Magnitue - Angle to Complexm 模块，它是把两个输入作为幅值和辐角来生成复数。图 5 - 20 给出了一个演示如何产生复数信号的示例模型，读者可以看到使用这两个模块得到的结果完全不同。此外，Math 子库还提供了一个把复数分解为实部和虚部的模块——Complex to Real Image 模块。

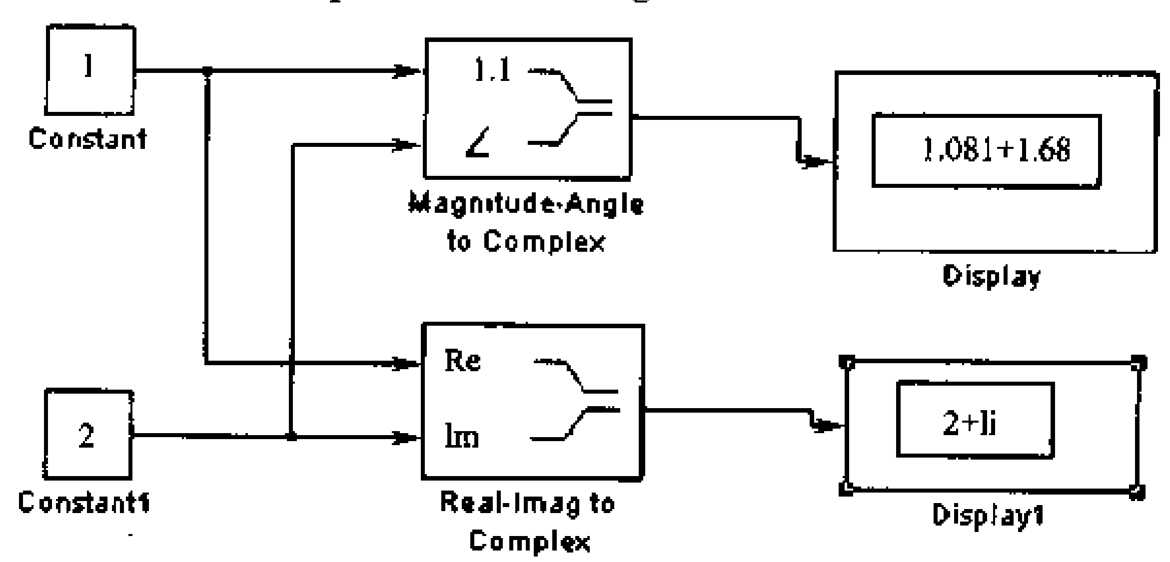


图 5 - 20 产生复数信号的示例模型

5.4 建立子系统

5.4.1 建立子系统

当模型规模很大、很复杂时，可以通过把一些模块组合成一个子系统，来简化模型。建立子系统有以下几个优点。

(1)可以减少显示在模型窗口的模块数，这样用户的模型窗口就会很整齐，也很方便用户；

(2)可以将功能相关的模块放在一起，用户可以用建立子系统创建自己的库模块；

(3)可以生成层次化的模型图表，即子系统在一层，组成子系统的模块在另一层。这样用户在设计模型时，即可采用自上而下的设计方法，也可以采用自下而上的设计方法。

在 Simulink 里创建子系统的途径有两种。

(1)增加一个子系统模块到你的模型，然后打开这个模块并在打开的模型窗口建立子系统；

(2)先把模块链接好，然后再把这些模块组合成子系统。

1. 通过子系统模块来建立子系统

这种方法首先往模型里加入一个称为 Subsystem 的库模块，然后再往该模块里加入组成子系统的模块，进行设计。

下面建立一个简单的子系统作为示例。该系统的功能是将输入的华氏温度转换成摄氏温度，读者可以在库浏览器的 Simulink extras 节点的 Transformation 库找到它的这个模块的原始版本。图 5 - 21 是建好的子系统。

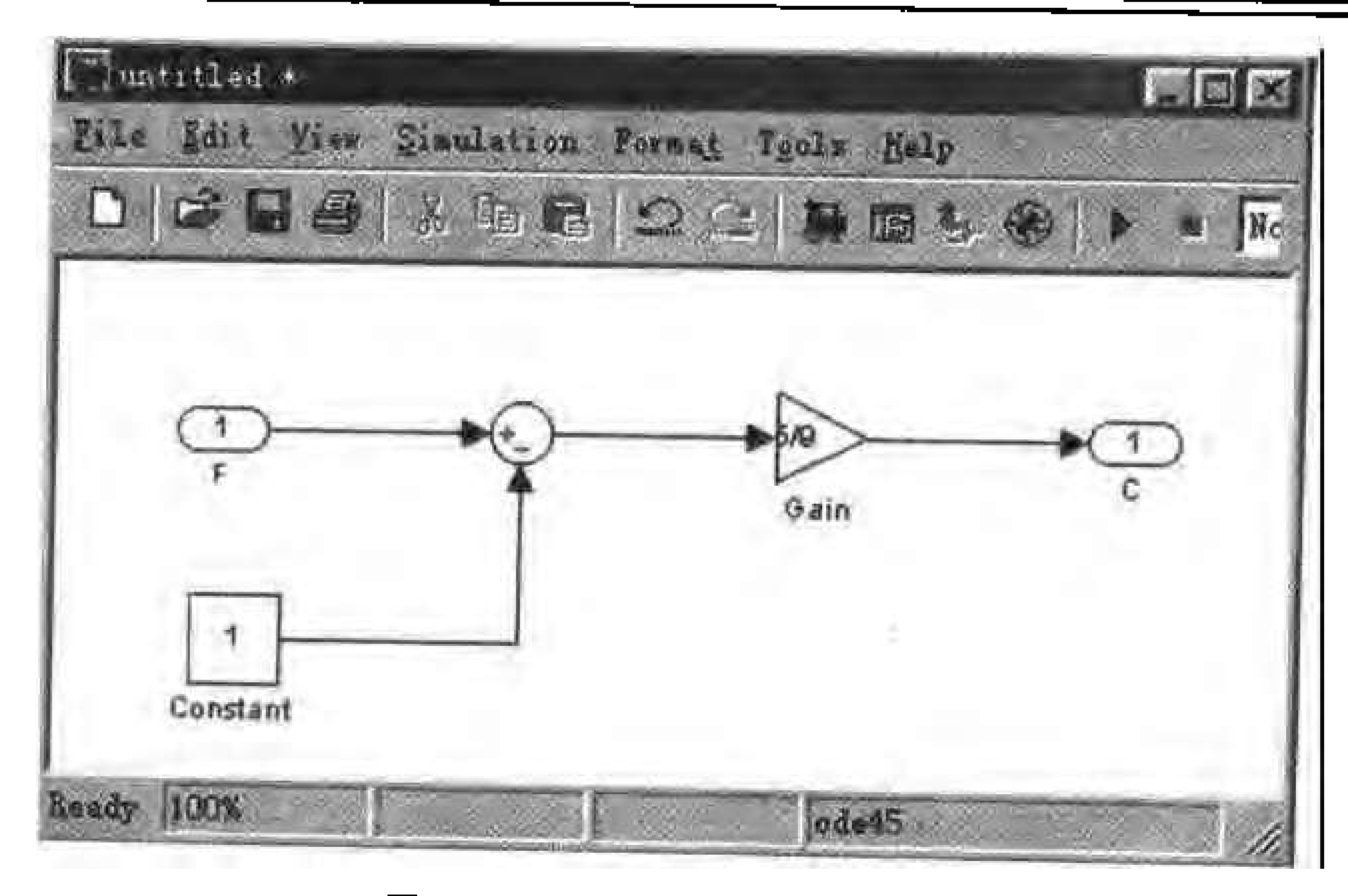


图 5-21　F2C 子系统及其内部结构

它的建立可以按下面的步骤进行。

(1)将 Subsystem 模块复制到模型窗口,它的位置在 Simulink 节点的 Singnal & System 子库里。

(2)双击 Subsystem 模块,这会打开子系统的编辑窗口,见图 5-21 的窗口。

(3)将组成子系统的模块添加到子系统窗口。为了定义子系统的输入、输出端口,必须手动地向子系统窗口添加 in1 和 out1 模块。这是两个虚模块,与它们相连的信号将作为子系统的输入输出信号。当然这只是 Simulink 图形化建模的一个逻辑方法,事实上在物理模型里,这是没有必要的,但是在 Simulink 建立子系统里它们是不可缺少的。

(4)按设计好的图表,连接好各个模块,并修改 in1 和 out1 模块下面的标签。实际上,所谓的标签就是这两个端口在子系统图标上的显示文字。

(5)关闭子系统窗口,把模型保存。读者可能想直接用子系统窗口 File 菜单里的 Save 命令来保存子系统,但是这个想法是行不通的。其实,这个命令也是保存整个模型,因为在 Simulink 里只有模型或库才是一个完整的独立实体。此外,重新定义子系统的名称是一个很好的习惯。

至此,就建立了一个子系统。

2. 组合已存在的模块来建立子系统

如果现有的模型里已经包含了用户想转化成 Subsystem 的模块,就可以通过组合这些模块的方式来建立子系统。

例如,现在已经有了一个如图 5-22 所示的模型。这个模型的功能是把离散信号累加。读者可以运行这个模型,to workspace 的功能是把存储仿真结果的输出变量保存到 MATLAB 的工作空间里。所以,要查看仿真结果,只需要在 MATLABA 命令窗口查看这个变量,这里是 simout。注意,这个变量是结构类型,结构内的字段存取在上一节讲过,不清楚的话,可以再回过头来复习一下。更详细的说明可以在 MATLAB 的帮助文件里找到。

读者会看到,Sum 模块和 Unit Delay 模块组合在一起完成了一个累加器的作用。下

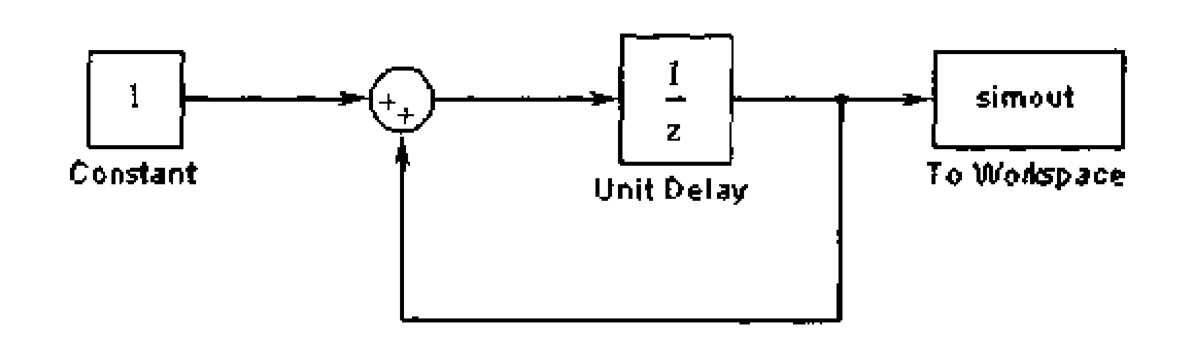


图 5－22　根据已有模型建立子系统

面将演示如何把它们转化为子系统。

首先按照图 5－22，建立好这个模型。其中，unit delay 模块和 to workspace 模块的位置分别是 Simulink 节点下的 Discrete 和 Sink 子库。

把已存在的模块转化为子系统，只要用 Simulink 用户界面 Edit 菜单下的 Creat Subsystem 命令即可。但是读者会发现此时的该命令处于不可用的状态，这是因为你还没有选择合适的操作对象。所以第一步操作应该是选取要组合成一个子系统的模块，这里别忘了连线也是对象之一。前面曾讲过如何在 Simulink 里选择多个对象，根据经验，第一种逐个选取的对象的方法在这里不能起作用，正确的方法是用方框包含待选择对象。正确选择好操作对象之后，Edit 下的 Creat Subsystem 命令就会变成可用状态。单击它之后，Simulink 会自动将操作对象转化成一个子系统，转化后的图如图 5－23 所示。

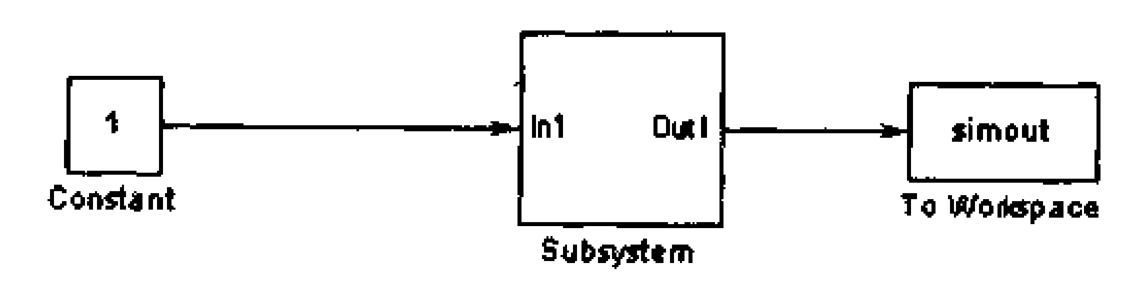


图 5－23　自动生成的子系统

读者双击生成的子系统打开子系统的窗口，将会发现自动生成的子系统内部结构和前面用手动方法生成的图表结构是一致的。也就是说两种方法的区别只在于输入输出端口的加入是自动还是手动。

这种方法生成的子系统美中不足的是端口名不直观，需要用户打开子系统窗口手动修改端口模块下的标签。模型里的标签必须是唯一的，因为 Simulink 把它作为此模块的名称，来标识模块，这一点在学习了模型文件格式之后将会更清楚。

5.4.2　用子系统来定义库

在 5.1.2 节曾提到过如何建立自己的库，但是那里所建立的库有一点不好，就是不能在库浏览器里浏览。这一节将介绍如何往库浏览器里添加节点。

例如，把前面生成的 F2C 子系统添加到 Simulink extras 节点的 Transformation 子库里。请你稍微思考一下，就能找到这种方法。假如 F2C 子系统已经建好，建立使用的方法上面讲到的两种都可以。首先，打开 Transformations 子库，方式是用右键弹出的菜单的“open”命令。然后用 unlock library 命令去除锁定，使该子库处在可修改的状态，再把已经建好的子系统 F2C 复制到 Transformations 子库窗口。操作还没有完全结束，因为以后

所建立的子系统里的模块有可能包含参考模块，也就是说它们不是独立的。而且，即使子系统内不包括参考模块，但是复制在 Tansformations 子库里的子系统其实是你前面建好的子系统的参考模块，也就是原来的子系统才是真正的库模块。这种情况的后果是当你存放初始子系统的模型或库有所变动时，存放在 Transhformaions 子库里的子系统就变成链接的模块，这显然是不符合库模块的独立存在性要求的。基于上述原因，一定要把复制在子库里的子系统的链接断掉，这一点可以用 Edit 菜单下的 break library link 实现。最后，保存改动后的库，就完成了添加。图 5－24 表示了这种情况后的结果。

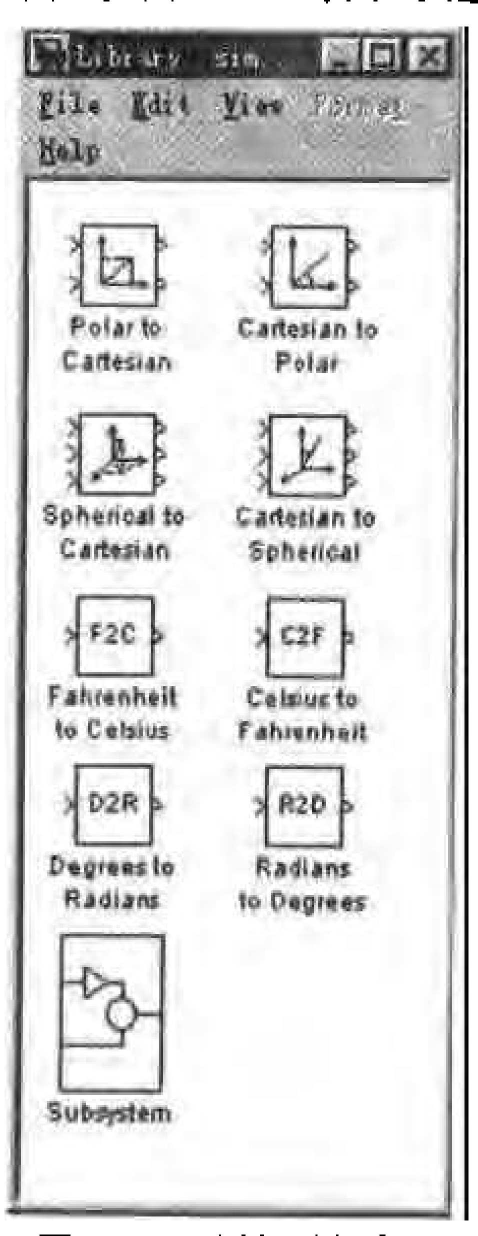


图 5－24 添加后的库情况

读者已经注意到了，添加进去的 F2C 在形式上更像是一个子库，而不像是一个模块。这是因为添加进去的本身就是一个子系统，而非一个模块，子系统的内部表图是可以浏览的。可以用封装子系统的方法，使它表现得像一个模块。关于封装子系统这个问题的详细信息将在下一节介绍，这里只对 F2C 进行简单的封装，用以演示封装后添加到浏览器的效果。

封装的方法也比较简单，请选择好初始建立的子系统，从 Edit 菜单找到 mask the subsystem 命令，这个命令也可以在右键弹出的菜单里找到。这将打开一个 mask edit 得对话框，请选择 document 页，在 blocks desription 编辑框里输入一行文字，比如“这是一个把华氏温度转换成摄氏温度的自定义模块”，按 OK 按钮关闭对话框。再按照前面讲过的步骤把它添加到 Transformations 子库。

这时，封装后的子系统已经表现得和库模块毫无二致了。

从图 5－24 可以得到另外一个启示，即库浏览器里，子库并不是一个库文件，确切地说是子系统。库文件由子系统构成，子系统里存放下一级子系统或者子库了。用户自定

义的子库通常放在 Simulink extras 库里，例如，可以把常用的模块放在一起组成一个“我的模块”子库(如图 5-25)。明白子库即子系统的道理，读者就不难完成这种添加，记住两点，一是要把子系统内的参考模块的链接去掉，二是不要封装子系统。

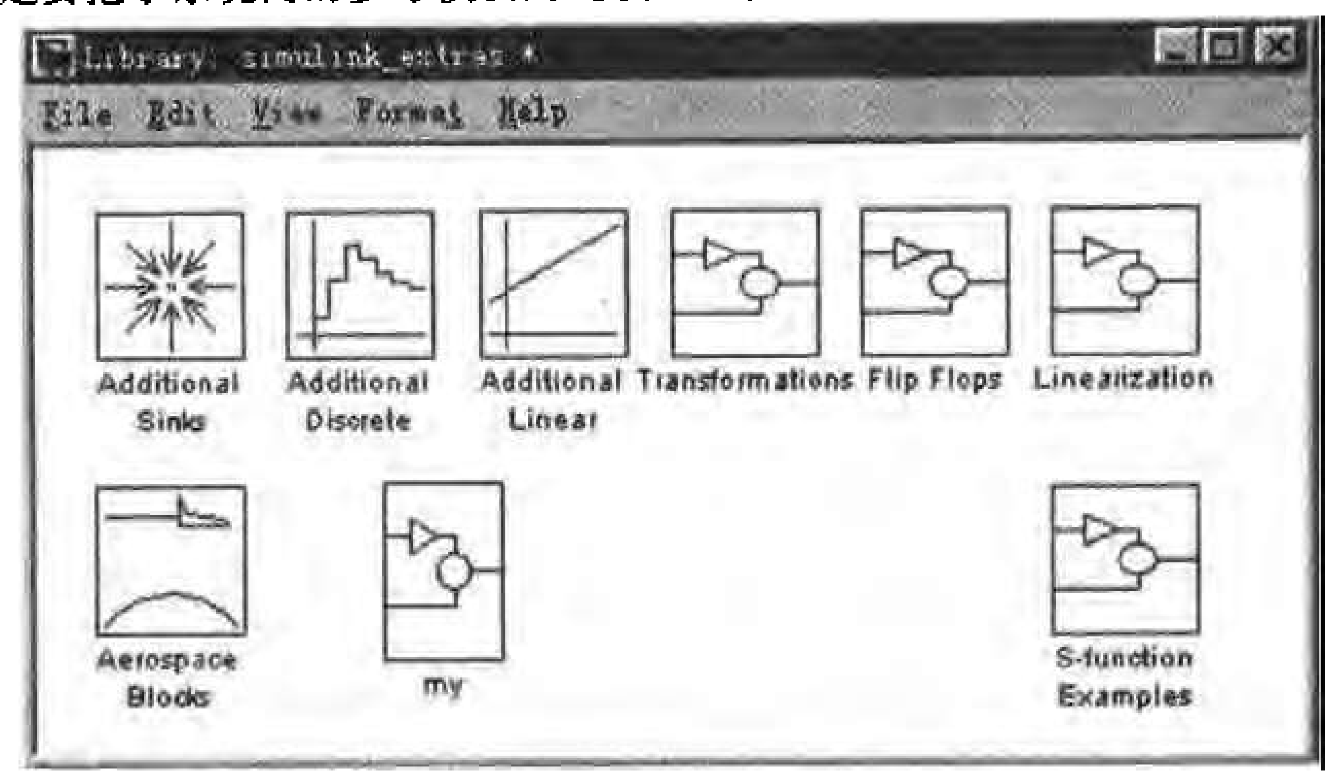


图 5-25 添加的“my”库

5.5 封装子系统

在前面一节里，我们略微提到封装一个子系统可以使子系统表现得和模块一样——双击后会出现一个参数设置对话框，这只是反映了子系统好处的一个方面。

封装是 Simulink 的一个十分重要的特性，它的使用使得用户可以自定义子系统的对话框体会图标。通过封装，用户能够：

(1)用一个子系统一个参数设置对话框代替多个对话框，以简化模型的使用。否则，使用模型时，就不得不逐个打开子系统内的各个模块的参数设置对话框输入参数。经过封装之后，这些参数就可以通过封装好的子系统对话框来设置，并传给封装过的子系统内的模块。

(2)定义一个具有用户自己的模块描述、参数字段标签和帮助文档的对话框，为模型使用者提供一个描述性更强、更好的用户界面。

(3)定义命令，计算那些取值依赖于模块参数的变量的值。

(4)定义一个能反映子系统目的，更有意义的模块图标。

(5) 把子系统的内容隐藏在定义好的界面下，避免使用者对子系统进行无法预料的改动。

(6)建立动态的对话框。

下面就来介绍如何对子系统进行封装。

5.5.1 子系统封装示例

图 5-26 是一个用以计算 y = mx + b 的简单子系统，其中的右图是该子系统的内部结构。它的内部模块有：一个增益参数为 m 的 Gain 模块，一个常数值为 b 的 Constant 模块，以及一个 sum 模块。在进入下面的讨论之前，读者先把这个子系统建好。

如果不进行封装，正如以前看到的，双击子系统会看到子系统的内部结构。为了改变

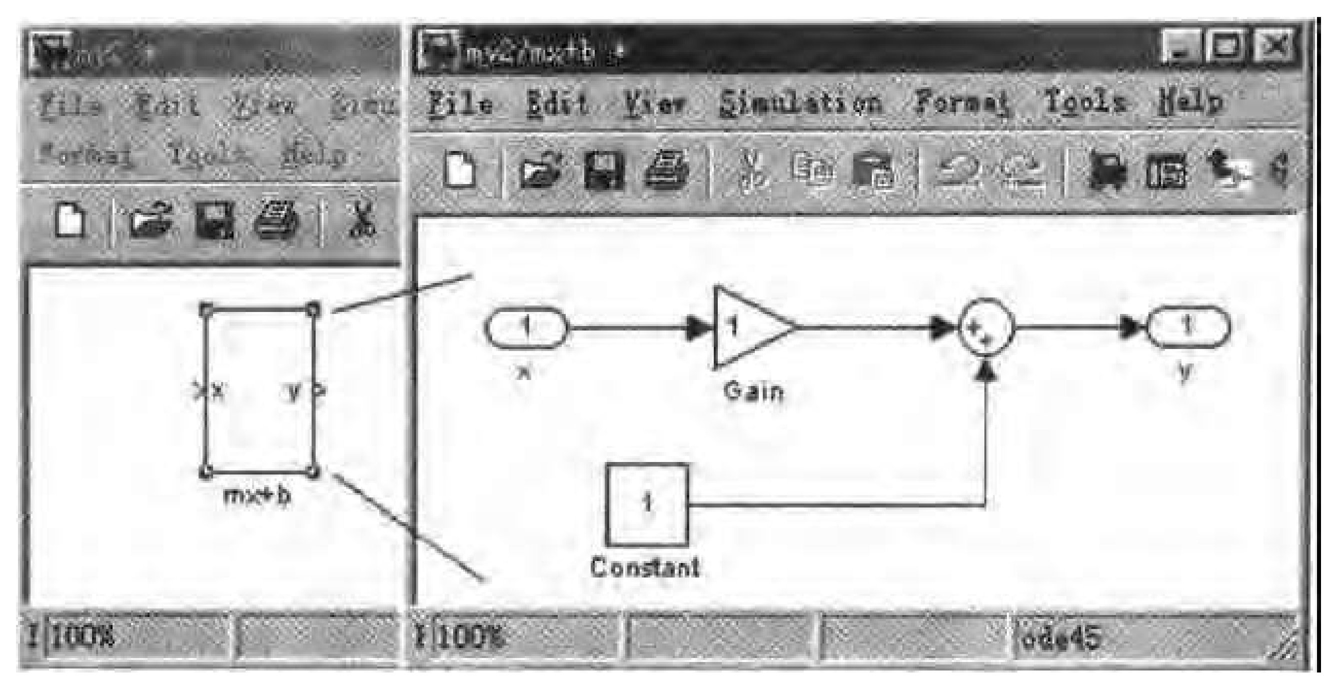


图 5-26 封装好的子系统

m 和 b 这两个参数，你必须分别打开 Gain 和 Constant 的参数对话框逐个设置，如果子系统内的模块很多，那么这项工作将是十分烦人的。

但是，封装可以简化这些操作。按照惯例，在开始讲解封装之前，本书先让读者体会一下，子系统封装后的不同凡响之处。如图 5-27 就是双击封装过的子系统弹出的对话框。



图 5-27 封装过的子系统的参数设置对话框

此时，设置 m 和 b 的值只需要打开一个对话框就可以了。

下面，就让我们来看看在 Simulink 里是如何封装子系统的。大体上说，一个封装要完成以下三件事：

(1) 定义提示对话框及其特性；

(2) 定义被封装的子系统的描述和帮助文档；

(3) 定义产生模块图标的命令。

1. 产生提示对话框

封装一个子系统，首先要选择该子系统，然后再从用户界面的 Edit 菜单选择 Mask Subsystem 命令。于是，就会出现读者前面已经见过的封装编辑器，它分为三页，出现在最前面的通常是 Initialization 页。图 5-28 是本例中该页呈现的样子。

正如在图上所看到的一样，用户可以定义的封装参数的属性有：Prompt、Type 和 Variable，其中，Prompt 是指用以描述参数作用的提示性文字。Type 是指用于输入或者

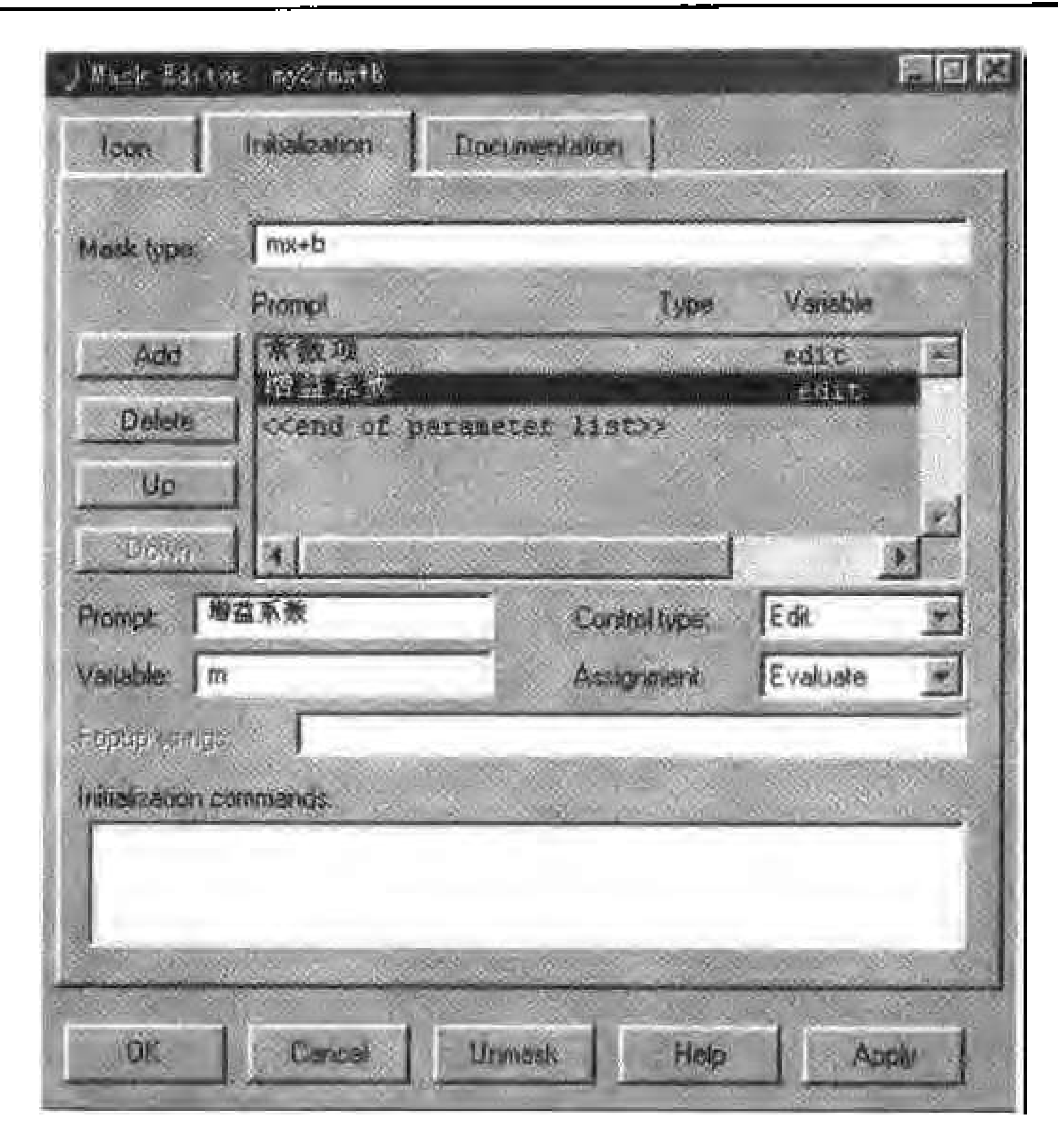


图 5-28 封装编辑器的 Initialization

选择参数的控件的样式，记住这里不是指参数的数据类型，比如在本例中选择了 Edit 控件。Variable 指存储参数值的变量名，也就是说用户在参数对话框输入的参数值将自动传给这些变量。实际上，这些变量存在于模块的封装工作空间——它是 Simulink 专门为模块分配的工作空间。它是用户定义变量的场所，只有它里面的变量才能被模块访问。但是，并不是所有的变量都只能是参数，只有像 b 和 m 这样在对话框里定义的变量才是需要用户设置的参数，实际上子系统里有很多模块的参数值都可以被预先设置好。读者可以从 Simulink 的许多 demo 里发现这样的现象，许多模块的参数设定为一个变量，但是找遍了整个模型，就是找不到给该变量赋值的地方，而模型却能够运行正确，其中的奥妙就在于使用了回调函数，本书的后面章节将会给读者讲解这一技巧。以上是题外话，下面还是接着封装的主题。

2. 建立模块的描述信息以及帮助文档

模块的描述信息和帮助文档是在封装编辑器的 documentation 页设置的。这一页的结构很简单，只有几个属性值，它们各自的功能读者很容易就能实验出来。图 5-29 描绘了它们与参数设置对话框的几个要素的对应关系。

3. 生成模块的图标

到目前为止，我们已经为 mx + b 定义好了一个自定义的参数设置对话框。然而，子系统模块依旧显示着模块的通用图标，这对使用者而言是很不直观的。一个比较好的替代图标是用一条斜线来表示。

模块图标可以在 icon 页定义。

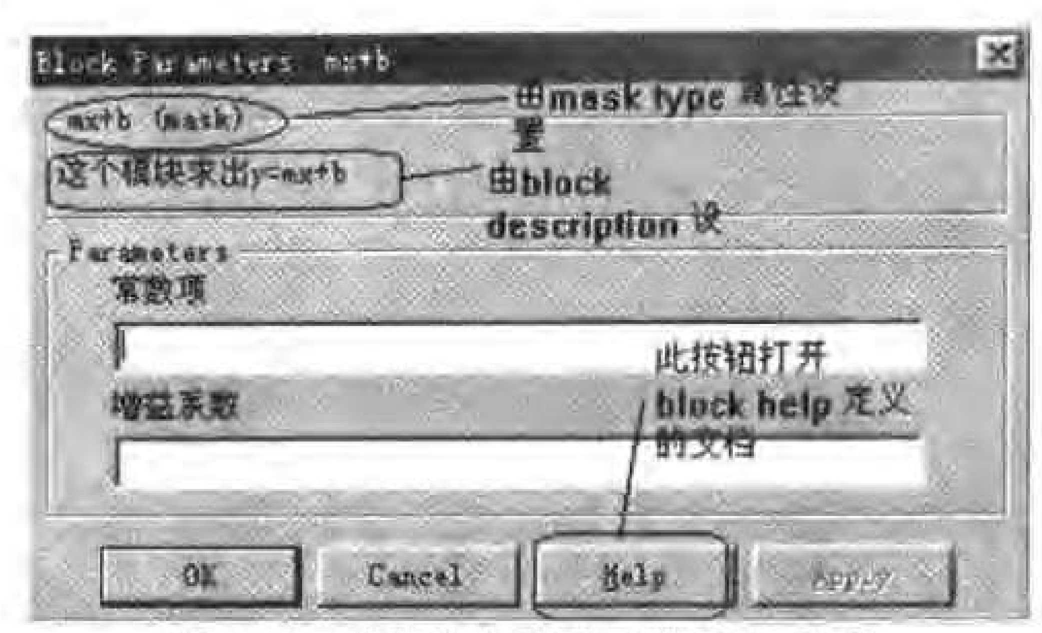


图 5-29　描述信息和帮助文档的显示位置

请在 drawing commands 提示下的 Edit 里输入命令：> >plot ([0,1],[0.,m]+(m<0)) % 这里“> >”依然表示是 MATLAB 命令。

上面用作图命令画了一条从点(0,0)到点(0,m)的直线，当 m<0 时，就将直线平移 1 以保证所画直线在模块框区域内。而且读者还可能注意到了在上述命令中，用到了变量 m，事实上，作图命令可以访问 mask 工作空间里的任何变量。这样随着参数 m 的改变，模块的图标也相应地改变，这就是一种动态图标。因为在用户还没有设置参数值时，m 依旧是个不确定的量，那么画图命令就无法运行了。

此外，还要把 icon 页上的 drawing coordinate 属性设置为 normalized。这样作图区域就被限制在左下角为(0,0)，右上角为(1,1)的正方形区域里。至于该页上的其他属性均不作任何修改，使用原来的缺省值，它们的含义在后面会具体介绍。

这样，再按 OK 按钮关闭封装编辑器就可以了。图 5-30 是封装后的图标样子。

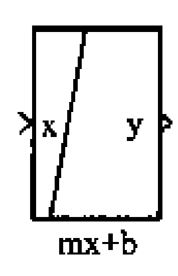


图 5-30　建立的图标样子

至此，我们封装示例就算完成了。下面将更深入细致地讲述封装编辑器。

5.5.2　initialization 页

在前面的一节里，这一页的大部分操作都已经讲过，所以在这里就不再详细说了。这里的注意力将会集中在前面有意回避的一些内容。

1. assignment 属性

assignment 属性的类型有两个值可以选择：evaluate 和 literal。如果 assignment 属性设置为 evaluate，那么模块使用者在模块参数设置对话框里输入的值，在被赋给变量之前，首先由 MATLAB 进行估值。如果 assignment 属性设置为 literal，则输入的值，不经过估值，而是直接作为字符串传给变量。为了弄清楚这两者的区别，请读者按图 5-31 建立一个测试模型。

实验的思路是：分别将 assignment 属性设置成 evaluate 和 literal，然后再输入相同的

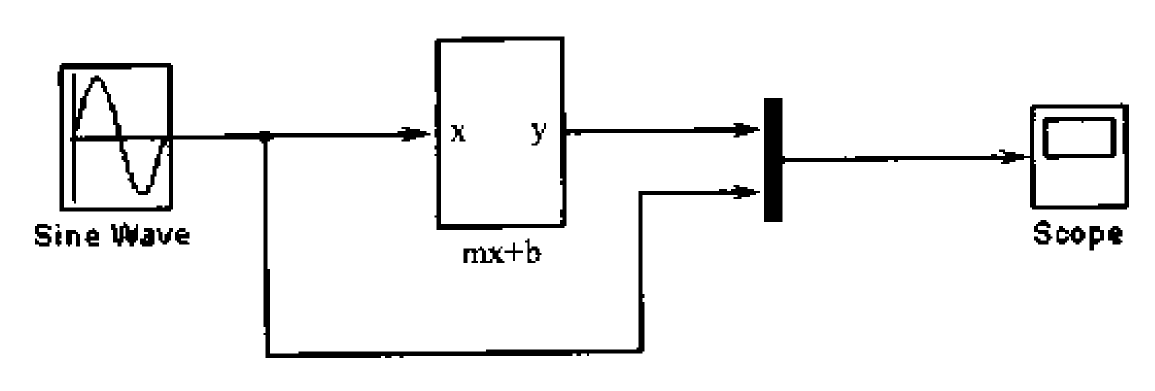


图 5-31　测试模型

字符串,用以对比两者的不同效果。

首先,用用户界面的 edit 菜单下的 edit mask 命令对 mx + b 模块重新封装,这里只是要把 assignment 属性设置为 evaluate。按 OK 按钮关闭后,再设置 m 和 b 的值,不妨设置 b 为 0,而在 m 的编辑框里输入字符串:gain,如图 5-32 所示。

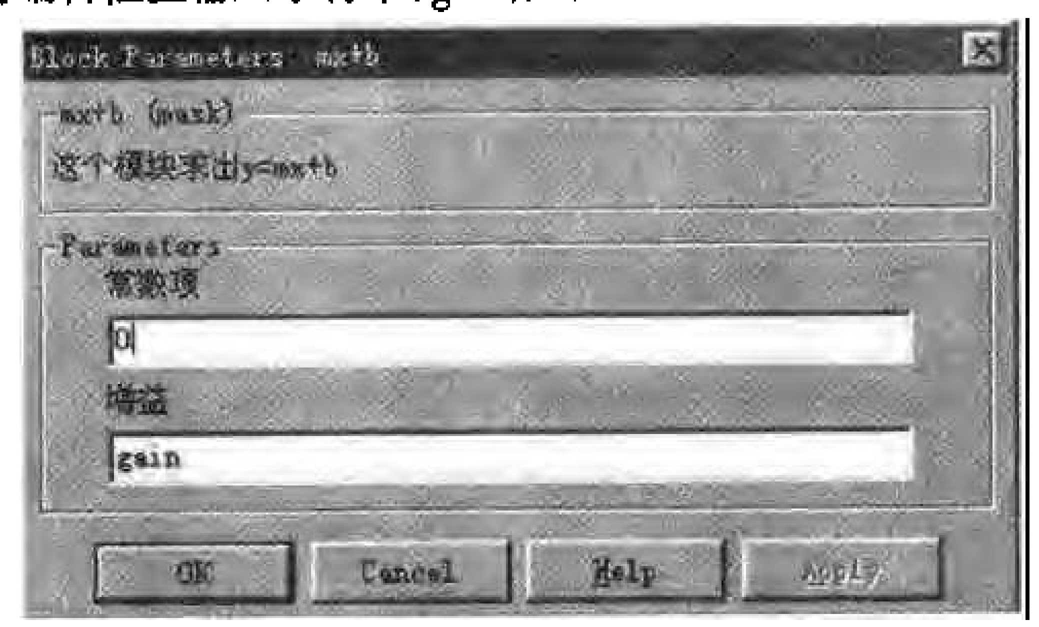


图 5-32　参数的设置

关闭参数对话框之后,运行仿真模型。这时,会出现一个错误提示,如图 5-33 所示。为了解决这个问题。请回到 MATLAB 命令窗口,输入

```
>> gain = 2;
```

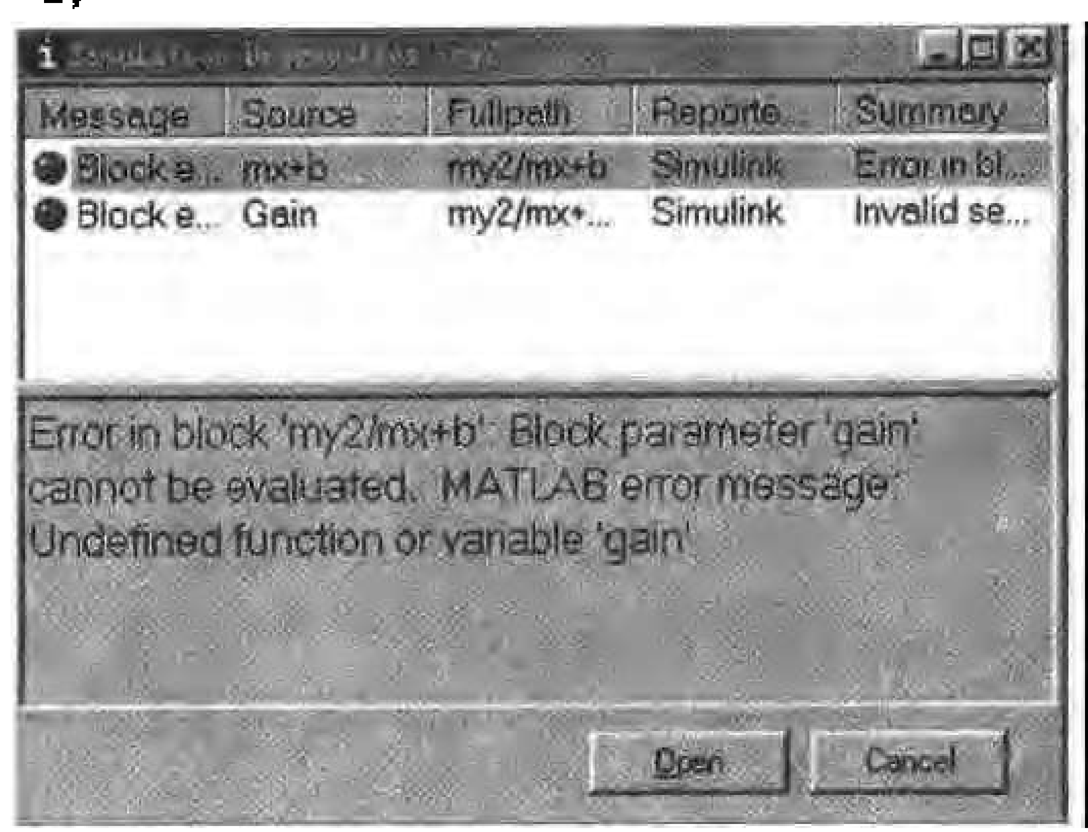


图 5-33　错误提示对话框

然后再运行仿真，这时就不再出现错误提示了，而且观察仿真的结果会及时发现，输出恰好是输入的两倍，也就是说，m 的值被设成了 2，恰恰是我们在 MATLAB 工作空间里为 gian 赋的值。这是怎么回事呢？

很简单，前面讲过，在对话框输入了 gain 之后，一旦开始运行仿真，就先由 MATLAB 进行估值，这就相当于在 MATLAB 命令窗口里输入了

```
>>gain
```

按照以前所讲的，MATLAB 对输入的变量估计顺序，首先它将检查是不是 MATLAB 工作空间里的变量，如果不是，再看是不是内置函数，如此下去，具体的顺序读者请看 MATLAB 有关静态仿真的章节。所以在 MATLAB 工作空间里定义 gain 变量之前，由于它不是内置函数，也不是搜索路径中的 mex 文件或者 m 文件，MATLAB 自然无法进行估量，于是就会出错。但是定义了 gain 变量之后，情况就不同了，MATLAB 把 gain 的值传递给参数变量 m，所以出现前面的仿真结果是很自然的，并不是偶合。

以此类推，在 m 的编辑框里输入一个函数名，也是可以的。读者不妨自己试试，例如 abs(-1)等等。

这些概念，在学习过程如何编写 S 函数之后，将是很显然的事情。

那 literal 又是怎么回事呢？

为此，请重新打开 mx+b 的封装编辑器，把参数 m 的 assignment 属性设置成 literal，别的属性的设置保持不变，然后重新在模块的参数设置对话框设置 m 的值为 abs(-2)。

```
>>abs(-2); %在编辑框里输入 abs(-2)
```

输入完后，请运行仿真模型。结果是可以预想的，Simulink 会给出一个错误提示。双击这些 Simulink 里的错误提示，Simulink 会告诉你错误在模型中的哪个位置，读者可以自己试试这些功能。

我们来分析出现错误的原因。因为封装中，参数 m 的 assignment 已经被设置为 literal 了，于是在参数设置对话框输入的'abs(-2);'MATLAB 并不对它进行估值，而直接当成字符串赋给参数变量 m，让模块自己去处理。于是 m 的值就是'abs(-2);'，当子系统里的 gain 模块访问 m 时得到的值就是'abs(-2);'，自然会出错了。改正的方法很简单，基本原理是由用户自己进行估值。为此，请读者再打开封装编辑器，在 initialization commands 里输入

```
>>m = eval(m);
```

eval 命令是用来对字符串进行估值的，更确切地说，eval 命令把输入的字符串，作为命令来执行。比如上面的 m 的值是'abs(-2);'，当使用 eval 命令时，它就是把字符串的内容 abs(m)当成命令来执行，然后返回执行后的结果，也即 m = eval('abs(-2);')，实际上就是 m = abs(-2)。

如此改动之后，再执行模型就不会出现错误了。

估值的位置也不一定非要出现在初始命令时，也可以是在子系统内部模块访问 m 变量时，即在 gain 模块的参数编辑框里输入

```
>>eval(m);
```

这样的效果是一样的。

一般，在实际使用时，literal 并不常用，通常使用的是 evaluate。

2. 控件类型

通过选择控件类型，用户可以决定是以直接输入的方式还是以选择输入的方式输入参数值。Simulink 提供了三种控件类型：editfields、checkboxes 和 popupcontrds。图 5－34 显示了一个使用三种控件的参数设置对话框。

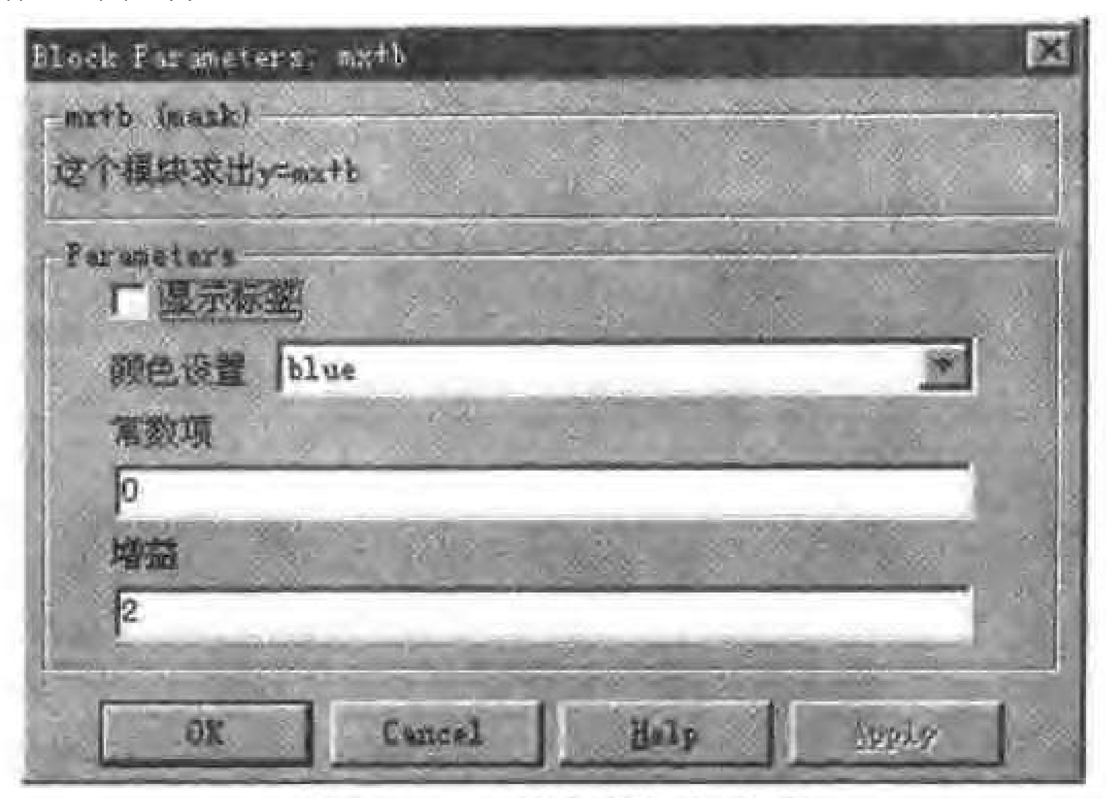


图 5－34　三种控件类型示意图

其中，editfield 控件是前面的例子中使用过了的，也是最常用的一种。设置参数时，用户没有必要直接在编辑框里输入要设置的值，正如前面所看到的一样。

在这里，要对前面关于 assignment 属性的一些论述加以修正。应该说前面所讲的原则大体上没有错，也就是估值和不估值的区别。但是前面的论述只是在控件类型是 editfield 时才是对的。表 5－6 描述了此情况下，参数是在不同的 assignment 属性下的赋值方式。

表 5－6　editfield 的变量取值与 assignment 属性的关系

Assignment	取　值
Evaluate	参数变量取值为输入表达式进行估值后获得的结果
Literal	参数变量取值为输入表达式的实际字符串

与 editfield 的直接输入不同的是，checkbox 只能让用户在选择 checkbox 和不选择 checkbox 两种备选项里选择。定义一个 checkbox 也没有什么特别之处，和 editfield 类似。但是和 checkbox 变量关联的变量的赋值方式不一样，它如表 5－7 所示。

表 5－7　checkbox 的变量取值与 assignment 属性的关系

Checkbox	Evaluate 方式下参数变量的取值	Literal 方式下参数变量的取值
选中	1	'on'
未选中	0	'off'

和前面两种方式相比，popupcontrol 略微有些不同。它的作用也是提供备选项让用户选择，但是它不像 checkbox 那样只有两个，它提供了所有可能的取值的列表。一旦 control type 选择了 popupcontrol，那么原来一直处于不可用状态的 popupstrings 编辑框将变亮，它是用来输入 popup 列表里的所有可能的取值项。不同的子项要用“|”来隔开。图 5－35是它的示例。



图 5-35 定义 popup strings

表 5-8 则是与 popup control 关联的变量取值与 assignment 属性的关系。

表 5-8 popup control 中变量取值与 assignment 属性的关系

Assignment	取 值
Evaluate	选项的索引值,索引值从 1 开始排列,例如,当第三个选项被选择,参数值为 3
Literal	标识选项的字符串,例如第三个选项被选择,参数值就为'Red'

3. 可调参数 (Tunable Parameterst)

所谓可调参数是指能够在运行时动态修改的封装参数。当你建立一个封装,它的所有参数都是可调的。但是用户可以通过模块的 MaskTunableValues 属性,来控制封装参数的可调与否。用户也可以为封装过的库模块定义缺省的参数值,操作的具体步骤是:

(1)解除库锁定状态。

(2)打开模块的参数对话框,在编辑框输入要设定的缺省值,然后关闭对话框。

(3)保存模型。

于是,当这个模块复制到模型中时,对话框打开,缺省参数值将会出现在模块的对话框中。

4. 初始化命令 (Initialization Commands)及其调试

初始化命令用于定义驻留在模块的封装工作空间内的变量,这些变量通常不希望模块使用者修改它,但是它又是必需的。在初始化命令里定义的变量能被子系统内的模块访问,也能被其他的初始化命令访问,还能被制作图标命令访问。总之,和封装的参数变量没什么区别。

初始化命令在以下几种时候被执行:

(1)模型被导入。

(2)开始仿真,或者是模型的图表被更新(指调用 update datagram 菜单命令)。

(3)被封装的模块被旋转。

(4)模块的图标被重画并且画图命令用到在 initialization commands 定义的变量。

初始化命令,可以是任何的 MATLAB 有效表达式,包括 MATLAB 函数、运算符以及在封装工作空间里定义过的变量。

初始化命令的调试,一般有三种方法。

(1)把每个命令后的分号去掉,使命令的执行结果能在 MATLAB 命令窗口显示。

(2)放置一个 kdyboard 命令到初始化命令之中,使得运行能被键盘操作中断,并且把控制权交给键盘。关于 keyboard 的详细信息,请查阅帮助。

(3)在 MATLAB 命令窗口,输入下面两个命令之一。

```
>>dbstop if error
>>dbstop if warning
```

前者的作用是一旦初始化命令产生错误，其运行将被中断，用户就可以检查封装工作空间；后者则是在发生警告时才能如此。详细信息，请看 dbstop 帮助。

5. 封装子空间

一旦模块的封装里（注意 Simulink 提供的模块也同样是经过封装得到的，但不一定是子系统），包含初始化命令或者定义了参数变量，Simulink 就为模块建立了一个局部的封装工作空间。

被封装的模块不能访问 MATLAB 的工作空间或者其他的封装空间，这一点就像前面讲过的 M 文件函数的局部工作空间一样。封装工作空间的内容包括封装参数和由初始化命令定义的变量，这些变量能被封装的模块访问，如果是子系统，那它能被子系统的所有模块访问。

把封装工作空间同 M 文件函数使用的局部工作空间进行类比，有助于加深对前者的理解。读者可以把在内部模块（针对子系统而言）的参数设置对话框里输入的表达式，以及在封装编辑框里输入的初始化命令或者制作图标命令，当成 M 文件函数的一行语句。而子系统的内部模块，就像 M 文件函数内要调用的其他 M 文件函数一样，内部模块的封装空间（如果有的话）和子系统的封装空间是不相同的，它们之间的联系仅仅是内部模块的模块参数。

在前面的 mx + b 模块中，m 和 b 作为定义好的封装参数被存储在 mx + b 的封装工作空间里。但是它们不会被自动地赋给子系统内的 Gain 模块或者 Constant 模块，而要这些模块来访问这些变量。正如 mx + b 模块封装内的命令，如初始化命令，不能使用在 MATLAB 工作空间定义的变量一样，gain 内部的命令也不能直接访问 mx + b 封装工作空间里的变量，而只能通过 Gain 模块自身的封装参数，如与'gain'提示文字相关的变量，来传入这些值。同样，mx + b 的封装工作空间也不能看到 gain 的封装空间定义的变量，这也就是封装的意义。图 5－36 可以大体反映这种关系。

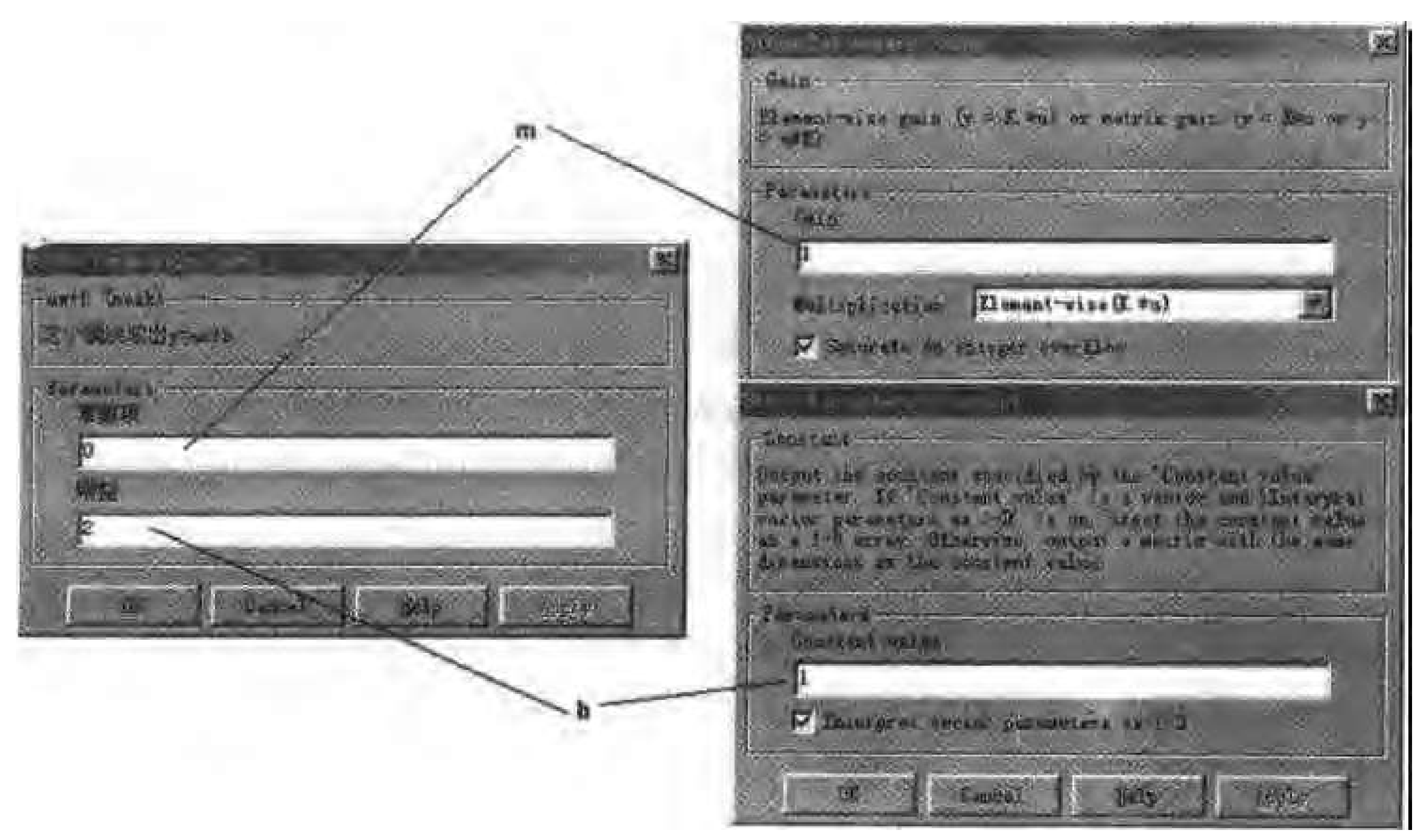


图 5－36　封装子空间

5.5.3 icon 页(图标页)

我们知道,icon 页的主要作用是让用户能够自定义模块的图标,它的内容相对简单,只有一个画图命令栏以及几个控制图标属性的 popupcontrols。这一小节将讲述如何产生不同形式的图标,包括:在模块图标上显示文字、在图标上显示图形、在图标上显示图像以及在图标上显示传递函数。最后还会讲到如何控制图标的属性。

1. 在图标上显示文字

定制图标的画图命令都是写在画图命令栏里的,画图命令访问封装工作空间的任何变量,这是前面讲过的内容。要在图标上显示文字,可以使用的命令有:disp、text、fprint 和 port _ lable。

它们的用法分别是:

```
disp('text')
disp (variablename)
text (x,y,'text')
text (x,y,text,'horizontalAlignment',halign, 'verticalAlignment', valign)
fprint ('text')
fprint ('format', variablename)
port _ label (port _ type, port _ number, label)
```

其中,disp 命令可以把字符串或者变量的内容显示在图标的中心,这里的变量的内容不一定非要是字符串,也可以是数组。

Text 命令的作用是,放置一个字符串(输入参数 text 或者 stringvariablename 的内容)再由输入参数(x,y)指定位置。坐标的单位取决于图标的画图坐标属性的设置。从上面的用法列表还可以看到:Text 提供了两个参数:'horizontalAlignment' 和 'verticalAlignment',分别来控制字符串相对于(x,y)的水平对齐和垂直对齐方式。例如,在 mx + b 模块的画图命令栏输入:

```
> > text (0.5,0.5,'mx + b','horizontalAlignment','center')
```

请读者仔细体会这种设置属性值的方法,先指定要设置的属性名——'horizontalAlignment',在紧接着设置它的值——'center'。这种方法在用命令设置模块属性时会经常使用,这样的好处是避免了繁琐的输入参数对应,就以 text 命令而言,可以不设置'horizontalAlignment'属性,直接设置'verticalAlignment',也不会造成混乱。

Text 命令支持的水平对齐选项和垂直对齐选项分别如表 5-9 和表 5-10 所示。

表 5-9 text 命令的水平对齐选项

选 项	对齐方法
Left	Text 的左端处于指定点
Center	Text 的中点处于指定点
Right	Text 的右端处于指定点

表 5－10 垂直对齐选项

选 项	对齐方法
Bcx	Text 的基线处于指定点
Botton	Text 的底线处于指定点
Middle	Text 的中线处于指定点
Cap	Text 的首线处于指定点
Top	Text 的顶线处于指定点

Printf 命令能够在图标的中心位置显示格式化了的文本(text 参数或 ariablename 的内容),文本的格式遵循参数“format”的设定。

要把文本分成几行显示,可以用‘ \ n’来表示断行。如图 5－37 所示。例如

> > disp ('SampleMask')

> > disp ('Sample \ nMask')

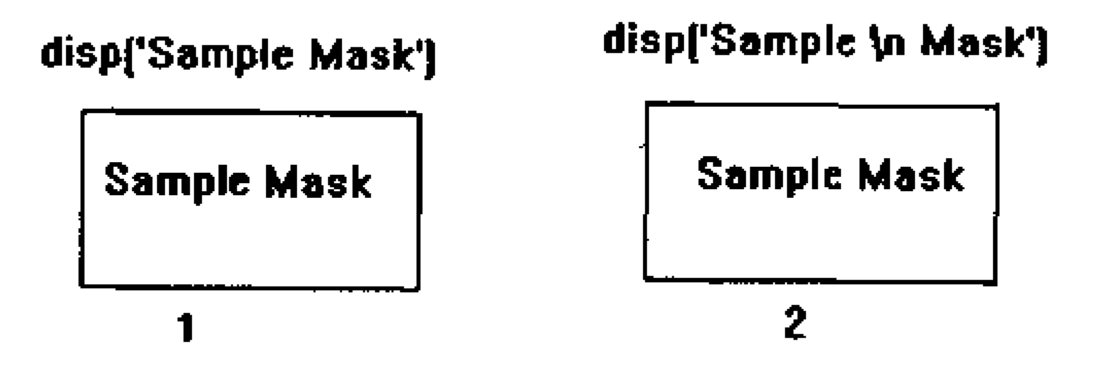


图 5－37 将文本分行

port _ label 命令可以让用户指定显示在图标上的标签名,这个命令的语法为:

prot _ label (port _ type, port _ number, label)

其中,port _ label 的值可以取“input”或者“output”,port _ number 是一个整数,label 说明这个端口的标签。例如,命令

port _ label ('input', 1, 'a')

定义了端口 1 的标签为‘a’。

2. 在图标上显示图形

这里的显示图形,实际上指为图标绘图。完成绘图功能的函数有 plot,它的形式有下面两种:

plot (Y);

plot (X1, Y1, X2, Y2, …);

关于 Y, X1, Y1, …参数的意义和 MATLAB 命令函数中的差不多。只是要注意,在绘制图标时,plot 不支持使用不同的颜色、线型和标记点的选择。如果用户试图使用这些功能时,例如

> > plot ([0 1 5], [0 0 4], ' r ')

这个表达式在 MATLAB 命令窗口输入,是没有任何问题的。但是在画图命令栏输入,则显示的图标是三个问号的形式。它们表示所使用的绘制命令出现了问题,可能是以

下几种情形：

(1)绘图命令用到的参数尚未定义(例如,当封装刚刚建立,还未在对话框输入参数的值)。

(2)模块参数或者绘图命令输入不正确。

3. 在图标上显示图像

用绘图命令绘制图形毕竟很简单,而且绘制过程还很麻烦。最方便的途径是直接把图像显示在图标上,Simulink 向用户提供了这种能力。

在 MATLAB 里用来显示图像的命令是 image。它要求输入的参数是一个存储图像 RGB 三色值的数组,而不是图像的文件名。因此,使用 image 之前,必须先把图像文件转化为 MATLAB 的一个数组。这就需要用到命令 imread 或者 ind2rgb,读取位图文件并将它转化成 image 命令必需的格式。

图 5-38 是把一幅图显示在模块的图标上的效果。



图 5-38 在图标上显示图像的效果

其产生方法是,首先在初始化命令栏里读入数据,并把它转化成 RGB 格式的数组。例如,要把 kids.tif 文件显示在图标上,可以这样写命令,如图 5-39 所示。

Initialization commands:

```
[x,map]=imread('kids.tif');
x=ind2rgb(x,map)'
```

图 5-39 在图标上显示图像的命令

其中,kids. tif 是 MATLAB 附带的一个图像文件可以在 toolbox \ image \ imdemos 目录下找到,由于该目录处于 MATLAB 的搜索路径,所以不需要把绝对路径写出来。

然后再在绘图命令栏输入：

```
> > image (x);
```

这里之所以要把前面的两个命令写在初始化命令栏里,因为很多函数,如这里的 imread 在绘图命令栏里无法识别。除了在本小节里提及的用于画图标的命令,其他的命令在绘图命令栏都是无法识别的,即使是最简单的赋值运算。读者可以自己体会这一点。

制作图标时,image 支持的其他使用格式还有：

```
image (a,[x,y,w,h])
image (a, [x,y,w,h],rotation)
```

4. 在图标上显示传递函数

Simulink 里,有许多用于处理离散信号的模块,它们的图标往往用对应的传递函数来表示,就像图 5-40 所示一样。

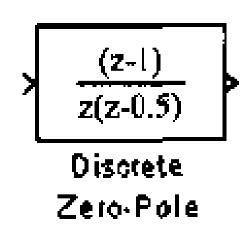


图 5-40　在图标上显示传递函数

这种图标显然不能用前面的方法得到。在图标上显示传递函数,可以使用下面的命令:

```
>>dpoly (num, den)
>>dpoly (num, den, 'character')
```

其中,mum 和 den 分别表示传递函数分子多项式和分母多项式的系数,属于向量类型;而"character"用来指定多项式表达式中的字母形式,通常缺省是"s",表示对应的模块是个连续系统。具体地讲,可以分为以下几种情形:

(1)按字母 s 的次数的降序排列,显示连续的传递函数。使用 dpoly(num,den)。

例如,num=[0 0 1], den=[1 2 1], 显示的图标为

$$\frac{1}{s^2+2s+1}$$

(2)按字母 z 的次数的降序排列,显示离散的传递函数。使用 dploy(num,den,'z'),还是用前面的 num 和 den,则显示为

$$\frac{1}{z^2+2z+1}$$

(3)把以上的按升序排列,只需把 s 和 z 变成 s- 和 z- 即可。

(4)按函数的零极点来显示,使用命令 droots。

5. 控制图标的属性

用户还可以设置一些参数来控制图标的属性,这些参数在 icon 页的右下端,设置的方式是从 popupcontrol 中选取一个选项。这些参数的作用解说如下:

(1)icon frame 指包含模块的正方形框,可以通过设置这个参数为 isible 或者 invisible,分别使得正方形外框可见或隐藏。

(2)icon transparency 参数用来设置图标的透明性,它有两个选项:Opaque 和 transparent。Opaque 表示图标是不透明的,这时它将会覆盖以前的 Simulink 自动在模块上显示文字和端口标签。反之,如果设成 transparent,则图标是透明的,不会覆盖端口标签等等。缺省设置是 Opaque。

(3)Icon Rotate 参数用来控制图标的朝向是否为固定,也有两个选项:fixed 和 rotate。当选择为 fixed 时,当模块被旋转或者翻转时,图标的方向保持不变;而设置为 rotate 时,图标将随着模块一起变化。也就是说,这两个属性实际上分别对应着绝对方向和相对方向保持不变。

(4)Drawing coordinates 参数的作用是调整图标绘制坐标。它的三个选项值意义分别是:pixel 表示按模块的实际大小调整模块方框的大小,autoscale 表示按所画图标的实际大小来调整模块方框的大小,而 normalized 则是把绘制图标的坐标系归一化。

5.5.4 documentation 页

封装编辑框的 documentation 页,是最简单的一页了,它的各个部分的意义十分明确。这里只着重强调两个地方:

首先,Mask type 这一项仅仅是为了文档说明的目的,没有特殊的意义。这和在建模时,模块下的标签不同,它是作为模块的名称存储在模型文件里的,是用来标识模块的,所以在模型里必须是互异的。这里的 Mask type 可以是为该封装选择的任何名称,它将会显示在封装后的模块对话框里。

其次,还请读者注意 Mask help text 这一栏。它主要是用来定义对话框的 help 按钮被按下后出现的帮助文档。它支持的格式有:

(1) URL 说明(以 http:、www、file:、ftp:等开始的字符串);

(2) Web 命令;

(3) Eval 命令;

(4) 显示在 Web 浏览器的静态文本。

要显示帮助时,Simulink 首先检查 Mask help text 的第一行,如果检测到 URL 说明、Web 命令或者 Eval 命令,Simulink 将直接访问这些命令定义的帮助文件;否则把 Mask help text 里的所有内容显示在浏览器里。

下面是几种能被 Simulink 接受的命令:

```
web ([docroot'/My Blockset Doc/' get_param (gcb, 'MaskType') '.html'])
eval ('! Word My_Spec.doc')
http://www.mathworks.com
file://c:/mydir/helpdo.html
www.mathworks.com
```

5.5.5 为封装的模块建立动态对话框

Simulink 支持用户建立随着用户输入参数变化的模块参数对话框。封装对话框的特性可以改变的方式,包括:

(1)参数控件的外观

改变一个参数可以导致另外一个参数对应的控制出现或者消失。而在一个空间出现或者消失时,对话框也会相应地扩张或者缩小。

(2)参数控件的使能状态

改变一个参数可以导致另一个参数的控件使能或者禁止输入。当一个控件被禁止时,它在外观上就变灰。

(3)参数值

改变一个参数的值可以导致使用相关联的参数被置为合适的值。

建立一个封装对话框必须将封装编辑器和 Simulink set_param 命令结合起来使用。具体地说,首先,要在封装编辑框定义所有的对话框参数,包括静止的和动态的。接着,在 MATLAB 命令行使用 Simulink 的 set_param 命令,来指定定义对话框对输入响应的回调函数(callback functions)。最后,就可以保存包含被封装的子系统的模型或者库来完成动态封装对话框的建立。

1. 设置封装模块对话框参数

Simulink 定义了一系列的封装模块参数来定义封装对话框的当前状态。用户可以使用封装编辑器来查看或者设置这些参数中的多数。同样可以使用 get _ param 和 set _ param 命令来查看和设置对话框参数。使用 set _ param 命令的好处在于,它可以在对话框被打开时设置参数,并立即改变对话框的外观。这也就能让用户建立动态封装对话框。

例如,读者可以在 MATLAB 命令行使用 set _ param 命令来设定自定义的模块参数被改变时,Simulink 要调用哪个回调函数来处理这种变化。而被调用的回调函数就依次地使用 set _ param 命令来改变封装对话框自定义参数的值或者它的状态,如隐藏(hide),显示(show)、使能(enable)或者禁止(disable)用户自定义参数控件。

2. 预定义封装对话框参数

Simulink 中与封装对话框有关的预定义参数有:

(1)MaskCallbacks

这个参数的值是一个字符串的单元数组,用来指定对话框用户定义参数控件各自的回调表达式。其顺序按照参数定义的顺序和单元一一对应,即第一个单元定义了第一个参数控件的回调函数,第二个单元对应第二个参数控件等等。回调可以是任何有效的 MATLAB 表达式,包含调用 M 文件的表达式,这就是用户实现复杂的回调。

为封装对话框设置回调最容易的方法是:首先,在模型或者库窗口选择相应的封装模块,然后,在 MATLAB 命令行输入 set _ param 命令。如下面的代码:

```
>>set _ param (gcb,'MaskCallbacks',{'parm1 _ callback',,'parm3 _ callback'});
```

这个命令定义了选中模块的封装对话框的第一个和第三个参数的回调函数。最后,别忘了保存包含封装模块的模型或者是库来保存回调设置。

(2)MaskDescription

这个参数的值是一个用于设定模块描述的字符串,它的作用就是设置它,可以动态地改变模块描述。

(3)MaskEnables

这个参数的值是一个字符串的单元数组,它定义了用户定义参数控件的使能状态。它的对应顺序和 MaskCallbacks 参数相同,即第一个单元对应第一个参数,以此类推。On 代表这个控件可以被用户输入;off 代表控件被禁止。于是用户可以在回调中动态地使能或者禁止用户的输入。例如,下面的命令

```
>>set _ param (gcb,'MaskEnables',{'on',on,'off'});
```

这个命令在封装对话框一旦被打开,就会禁止第三个控件。

(4) MaskPrompts

这个参数的值是一个字符串的单元数组,它用来指定用户定义参数的提示,第一个单元对应第一个参数,以此类推。

(5)MaskType

这个参数的值是与对话框相关的模块的封装类型。

(6)MaskValues

这个参数的值是一个字符串的单元数组,用以指定用户定义参数的值,对应顺序同上。

(7)MaskVisibilities

这个参数的值也是一个字符串单元数组，用以指定用户定义参数控件的可视与否，对应顺序同上。On 表示相应的控件是可视的；off 表示控件是隐藏的。例如：

> >set _ param (gcb,'MaskVisibilities', {'on','off','on'});

5.6 建立条件子系统

条件子系统是指它的执行受输入信号的控制的那一类子系统。在条件子系统里，控制子系统是否执行的信号被称为控制信号。进入子系统里的信号被称为控制输入。

条件子系统在建立复杂的模型时非常有用，因为这些模型的某些组件要受模型其他的组件控制的。例如，数字电路里的计数器，就是在每个时钟到达时刻计数器的值增加一个。像这样的受控子系统，在数字电路里还可以找到很多。所以说，建立条件子系统在实际建模中经常会遇到。

Simulink 支持三种类型的条件子系统：

(1)使能子系统。当控制信号为正时，它才执行。执行时间从控制信号从负变到正(过零点)的时间开始，并在控制信号为正的时间内持续执行。它也就是数字电路里所谓的电平触发。

(2)触发子系统。在发生触发事件的时刻执行。触发事件在 Simulink 里是用触发信号的上边沿和下边沿来表示的。

(3)触发使能子系统。当控制信号为正时，如果触发事件发生，则子系统执行。

5.6.1 使能子系统

使能子系统在控制信号为正值的时间步里执行。每一个子系统只有一个控制输入，它既可以是标量信号，也可以是向量信号。当为标量时，只要该信号大于零，子系统就开始执行。若控制输入为(注意控制输入和控制信号的区别)向量时，只要是其中一个信号大于零，那么子系统也执行。

Simulink 在判断是否执行时，主要采用的是过零检测技术(将在后面章节中提到)。只要检测到了过零点，并且斜率为正，则子系统开始执行。如果检测到信号滑过零线，且斜率为负的，则子系统停止执行。

生成一个使能子系统的方法很简单。只要把一个 enable 模块复制到子系统模块中即可，它的位置是在 Simulink 库的 signals&systems 子库里。图 5-41 是添加了 enable 模块后的子系统的外形。

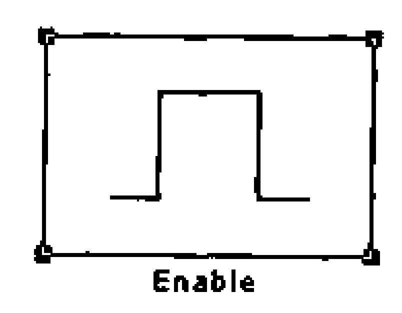


图 5-41　使能子系统图标

其余的操作和生成一般的子系统区别不大，或者说在这里 enable 更像是一个标记符号，它告诉 Simulink，该子系统将受到从这个端口输入的信号的控制。至于怎么控制，由 Simulink 来

处理，用户不需要考虑如何设计实现这一点。所以 Enable 模块在子系统的图表上是一个独立的，与别的模块没有任何的连线。用 Simulink 的术语讲，Enable 模块就是虚模块。

以一个半波整流系统为例。它的作用是在输入信号为正时，输出原信号，而在信号为负时输出为零。设计这个系统，就没有必要去考虑如何判断输入信号的正和负，使用 Enable 模块很容易就能做到这一点，只是这里输入信号和控制信号是同一个信号罢了。模型图表如图 5－42 所示。

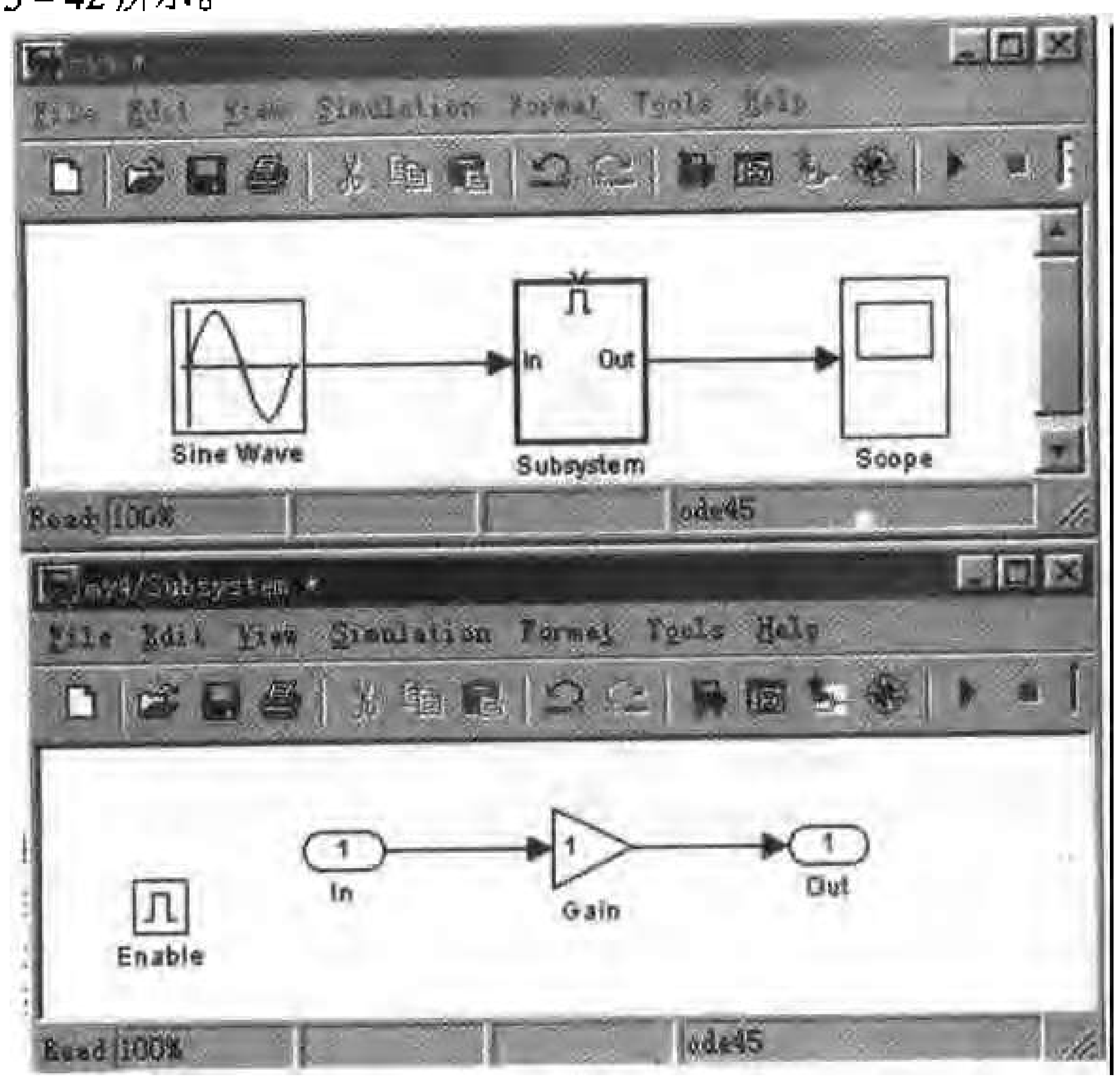


图 5－42 半波整流模型框图

经过上面这个例子的切身体会，读者对 Simulink 里建立使能子系统的步骤应该有了初步的了解。但是，使能子系统和普通子系统还是有些不同的。

由于使能子系统在控制信号为负值时，是不执行的，所以设计子系统时要设置当子系统不执行时的系统输出。为此，设计时要设置系统输出端口的参数。请双击打开输出端口的参数对话框，如图 5－43 所示。

当 Output when disabled 参数选择为 held，则该端口的输出保持最近时刻的输出值；如果为 reset，则输出被置为它的初始输出。而 initial output 参数定义了这个初始值。

在设计条件子系统中，另外一个要考虑的问题是，在系统被禁止时，是让系统保持上次执行的状态值，还是重新设置为系统的初始状态？这里，关于状态的概念将在后而章节中讲解。

为此，用户需要设置 Eanble 模块的参数。图 5－44 显示了 Enable 模块的参数对话框。图中，选项 hdd 和 reset 的意义与前面是类似的。上面 show output port 检查框的作用是允许系统输出控制信号，因为在某些场合下，控制信号还有别的用途。

图 5－43 输出端口参数对话框

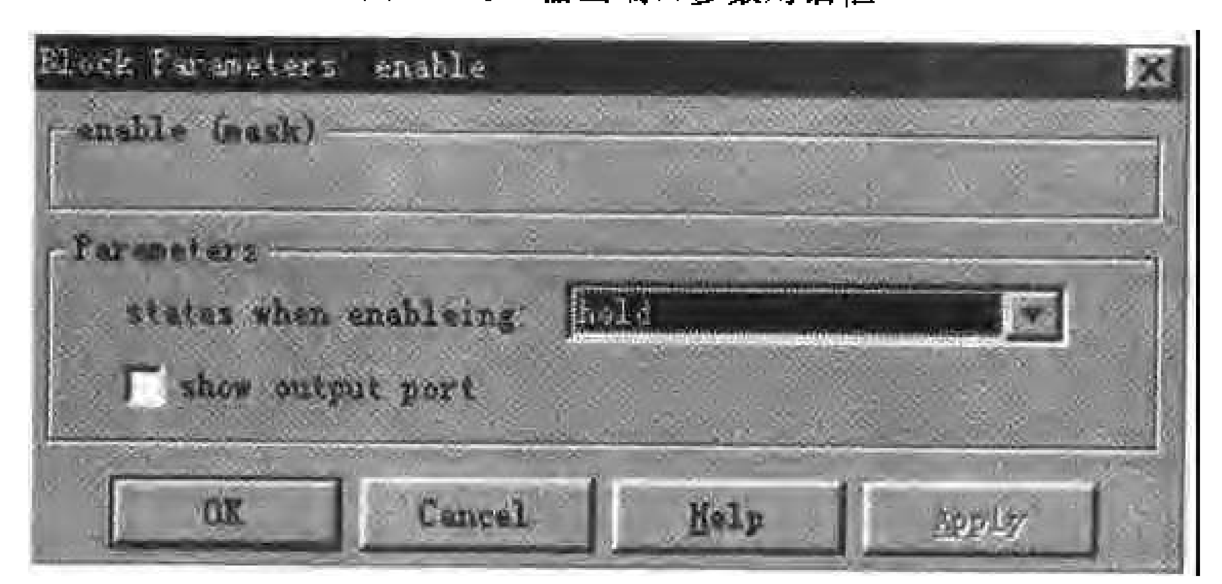


图 5－44 Enable 模块的参数对话框

除了上面的一些要求外，使能子系统可以包含任何模块。

5.6.2 触发子系统

触发子系统只在触发事件发生的时刻执行。触发事件是由控制输入信号的状态来决定的，一个触发子系统只能有一个控制输入，它在 Simulink 里被称为触发输入。

触发事件有两种类型：上升触发和下降触发。所谓的上升触发事件，指控制信号从负值或者零变为正值，也可以是从负值变为零。而下降触发事件，则是指控制信号从正值变为零或者是负值，或者从零变成负值。Simulink 允许用户选择子系统是在上升触发事件发生时执行，还是在下降触发事件发生时执行，或者是两者之一发生时执行。

例如，下面的控制信号，它的触发事件发生示意图为图 5－45，其中 R 表示上升触发事件，F 表示下降触发事件。

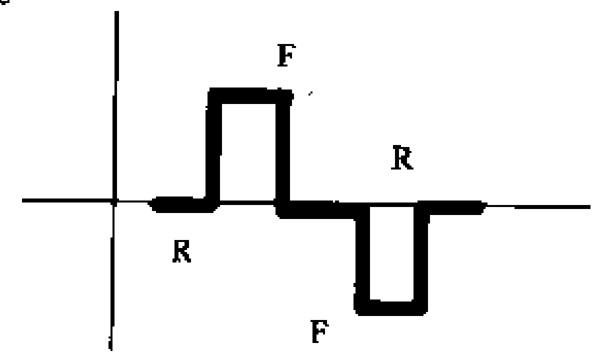


图 5－45 触发事件示意图

至于 Simulink 是如何检验触发事件是否发生，本书在后面章节将会讲到。

和建立使能子系统一样，在 Simulink 里，建立触发子系统的过程也是十分简单的。类似的，只要向子系统内加入一个 trigger 模块就可以了，它的位置也是在 Simulink 库下的 signal&system 子库里。图 5-46 是一个简单的触发子系统。而右边的小图则是触发子系统的内部结构。

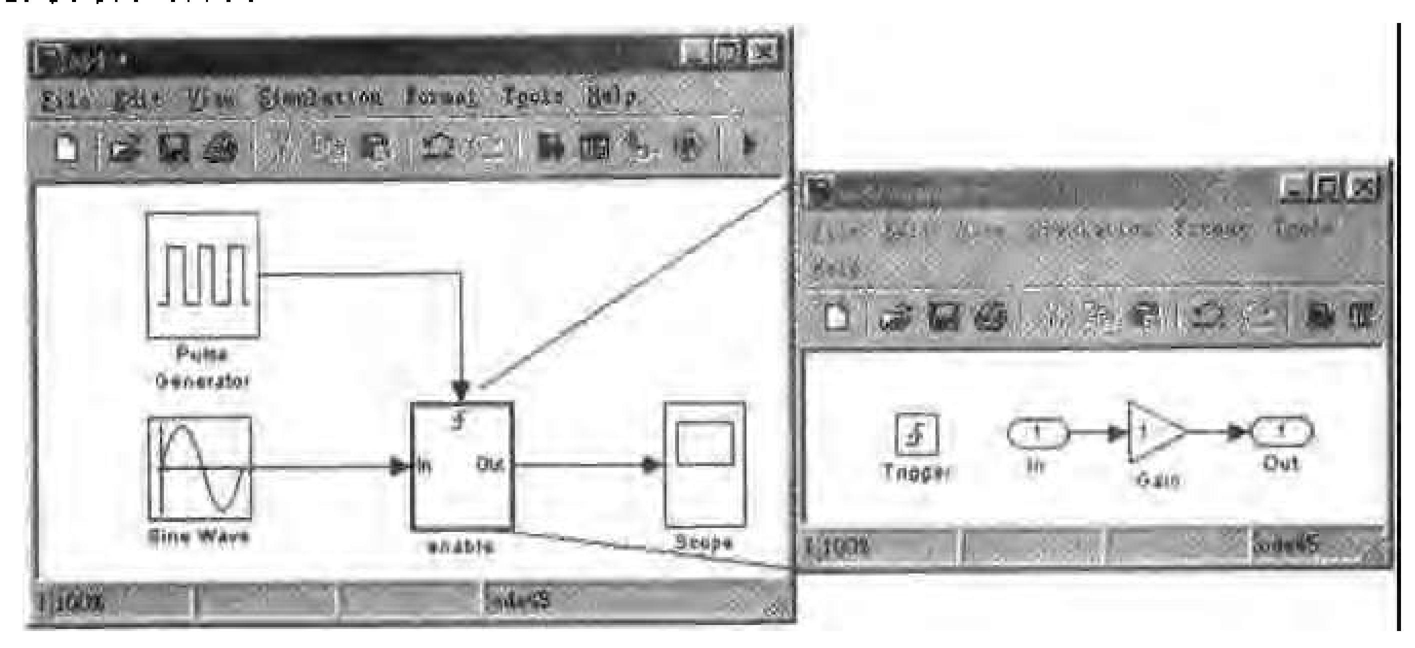


图 5-46 简单的触发子系统示例

读者可以仿照图 5-46 建立这个模型，其中 puls generator 在 source 子库里，它在这里产生一个时钟基准信号。在运行模型之前，请把 Unite delay 模块参数对话框的 sample times 改成 -1，它表明该模块的采样时间是继承的，受驱动模块的控制。关于继承的具体意义，请见后面的章节。

不难看出，在这个模型里，触发子系统起到了一个离散保持采样器的作用。为了把结果看得更清楚，可以用一个 mux 模块把正弦波、脉冲信号和模块信号合成在 scope 上显示。显示的结果如图 5-47 所示。

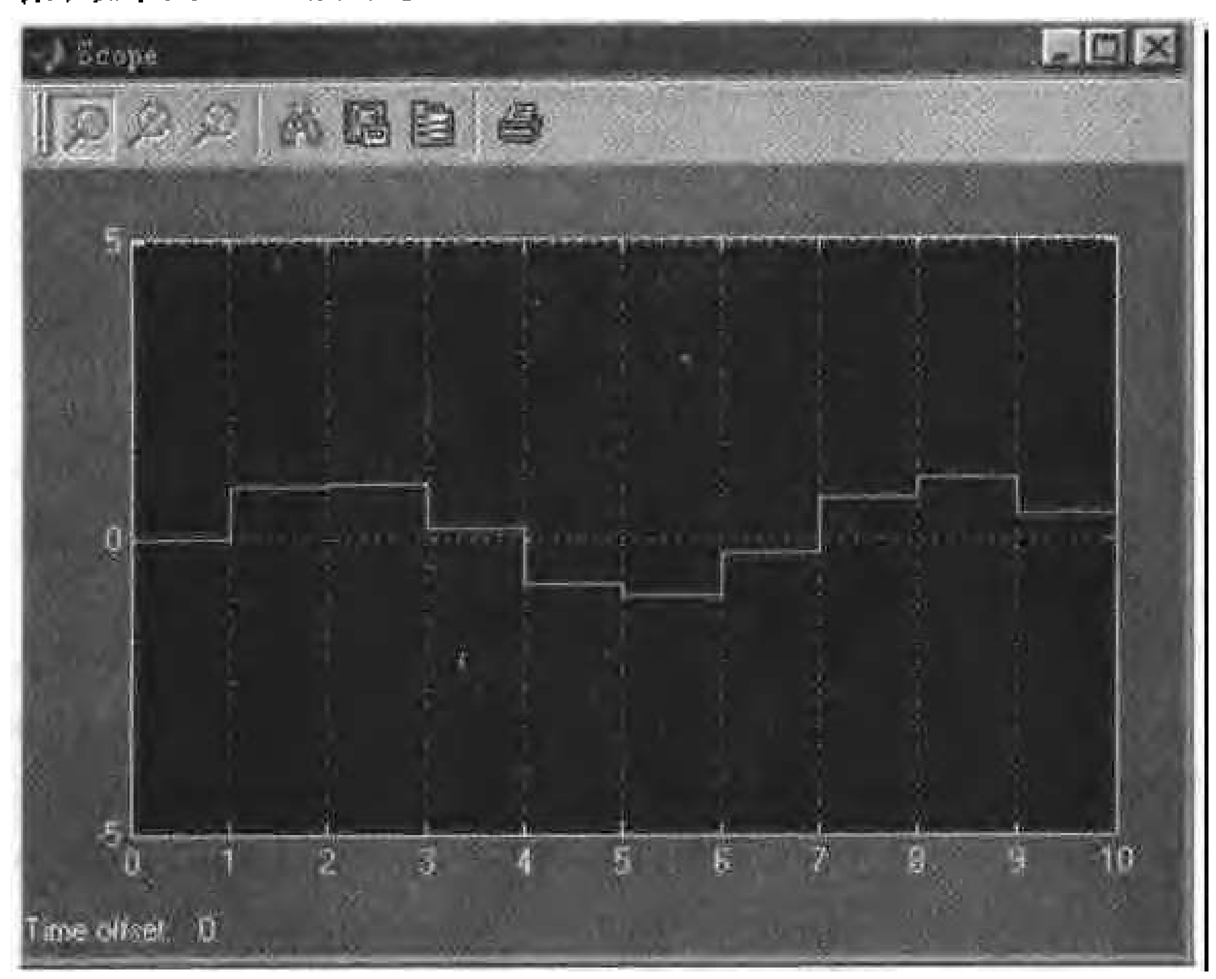


图 5-47 触发子系统运行结果

同使能子系统不同的是,触发子系统在两次触发时间之间,通常是保持最近的输出和状态的,即处于保持状态。读者在设置输出端口的参数时,会发现 output when disabed 这一参数无法设置。而 trigger 模块的参数对话框里根本就没有类似与 enable 模块的设置状态保持与否的项。在 trigger 模块里,允许用户设置模块的触发类型:rising、falling、either 和 function call。前面三个的意义是十分明显的,分别指在子系统在上升触发事件、下降触发事件和两者之一发生都执行。前面的例子,就是上升触发类型的。读者可以自己尝试一下不同的事件类型对仿真结果的影响。至于 function call,这里只是略微提一下,它与 S 函数的编写有关。简单地说,函数调用子系统,是指该触发子系统的执行不是由控制信号来决定,而是由一个 S 函数的内部逻辑来决定。什么是 S 函数,请看 S 函数一章。

触发子系统只能在仿真的特殊时刻才执行,因此,对能包含在触发子系统里的模块有一些要求:

(1)那些具有继承的采样时间的模块,如 Gain 模块或者逻辑运算模块;

(2)采样时间被设为 -1 的离散模块,这表示它的抽样时间从驱动模块继承过来。

5.6.3 触发使能子系统

使能子系统和触发子系统的结合,被称为触发使能子系统。可以从名称的字面意思来理解这种子系统的行为,在触发使能子系统里,使能的不是系统的执行,而是对触发功能的使能。图 5-48 是它的流程图。

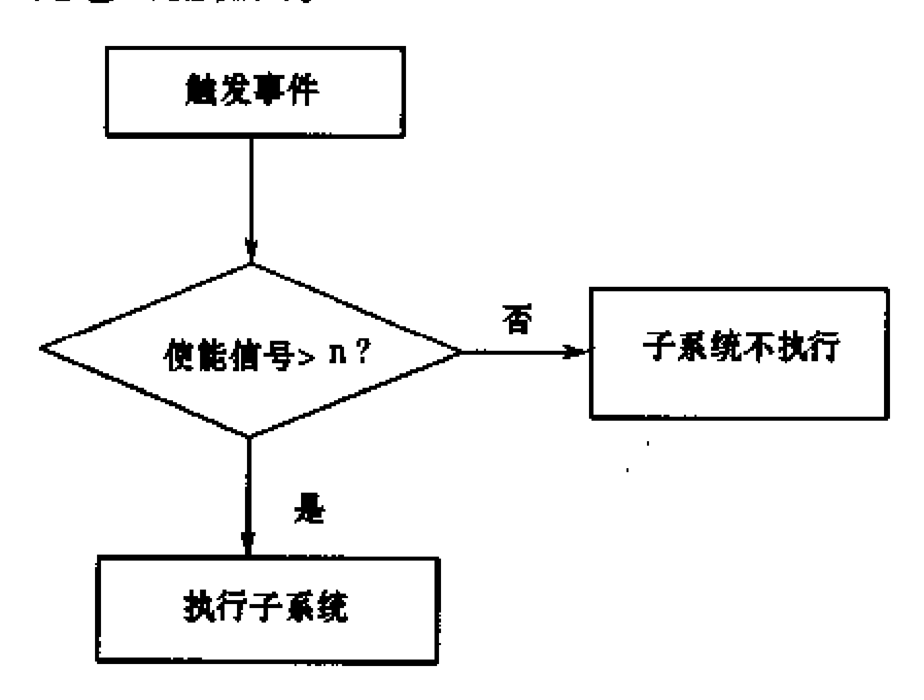


图 5-48 触发子系统流程图

在触发使能子系统里,触发控制信号和使能触发信号是分开的。当触发事件发生时,Simulink 就检查使能端的控制输入信号,如果它大于零,Simulink 就执行子系统。当两个输入信号都是向量信号时,只要向量信号有一个元素非零时,子系统就执行。

建立一个使能子系统,只要把 Enable 模块和 Trigger 模块都添加到子系统里就可以了。因为这里有 Enable 模块的存在,所以在建立触发使能子系统时,也要像建立使能子系统一样,设置当子系统在禁止时的输出和状态。实际上,在触发使能子系统里,Enable 模块和 Trigger 模块相互独立,所以它们的设置互不干扰。因此,可以分别按前面讲的方法对它们进行设置。

前面举过一个半波整流的例子,下面将来看看实现全波整流。最基本的思路是:让两个子系统在输入信号大于零和小于零的两种情况下分别执行,然后把它们的结果合并起来。

对于大于零的输入情况直接使用前面的半波整流子系统，而对于信号小于零的情况，只需把输入信号乘以一个 - 1，就可以利用前面的半波整流子系统了。在 Simulink 里，完成合并功能的模块是 Merge 模块，请注意这个模块和 Mux 的区别。图 5 - 49 是最后建好的模型。

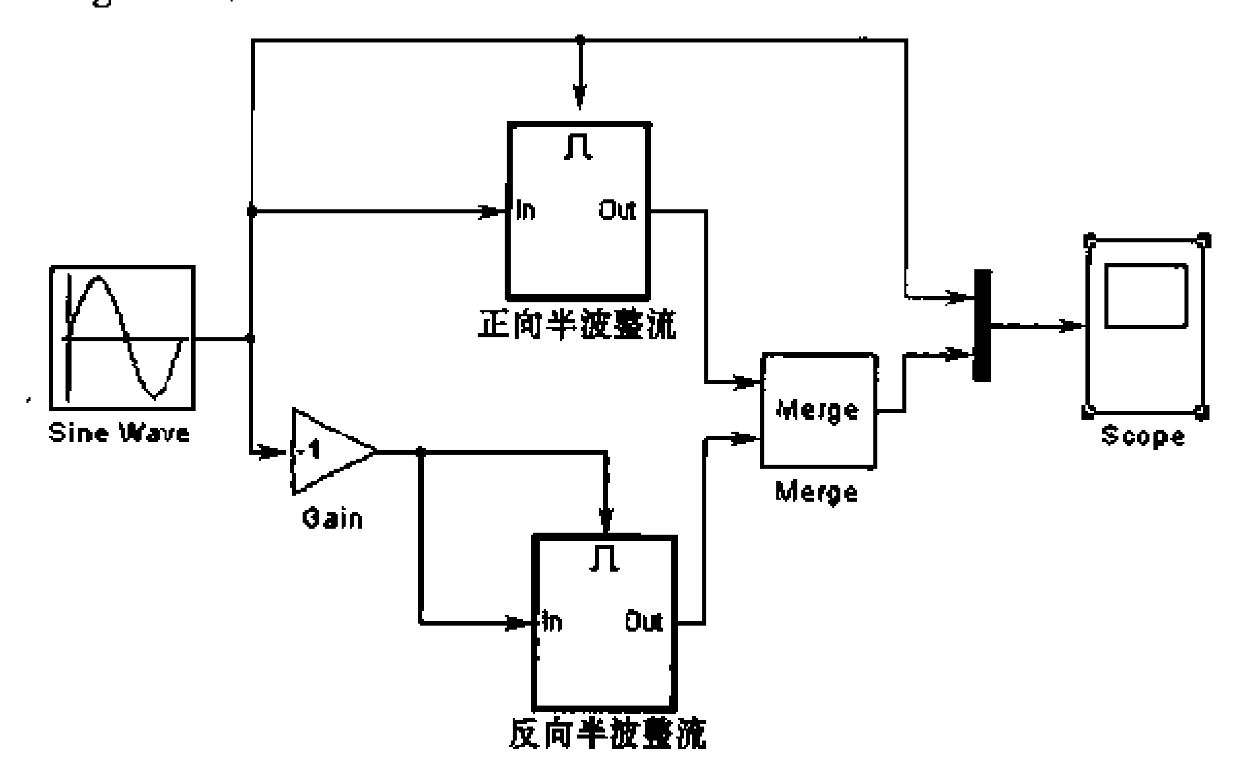


图 5 - 49 全波整流系统

其中 Merge 模块是把两个信号合成一个信号，它与 Mux 不同，Mux 是一个虚拟模块，只是把多个信号合并成一个 size 更大的向量信号，或者说只是空间上的合并。而 Merge 模块则是把多个信号在时间上合成一个信号，新信号在不同时间上的取值来自不同的源信号，也就是常说的时分复用，这是一个非虚拟模块。读者可以比较它们的输出，来体会这种区别。而图中的正向半波整流和反向半波整流都是和前面提到的半波整流的内部结构相同的。

运行模型后，输出结果如图 5 - 50 所示。

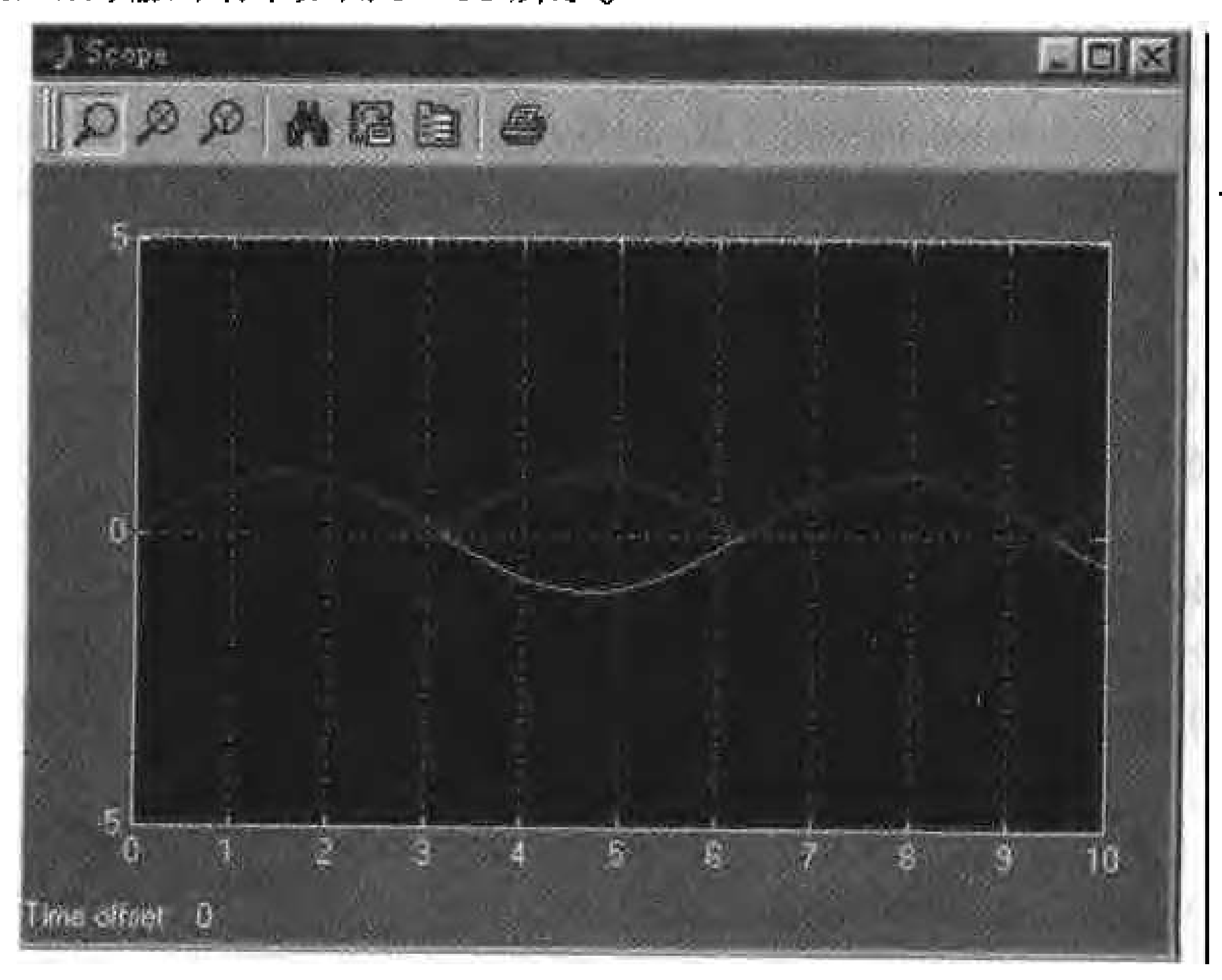


图 5 - 50 输出的结果

第 6 章　Simulink 调试器

无论你多么熟练，多么聪明，也不可能在建立模型时一点都不出差错。正像许多高级程序语言一样提供了功能强大的调试器，Simulink 也提供了一个工具便于用户查找和诊断模型中的错误。它允许用户通过单步运行仿真和显示模块的即时状态、输入和输出，来查明错误。这一章我们就来学习如何使用 Simulink 调试器来诊断 Simulink 中出现的问题。Simulink 调试器相对于高级编程语言的调试器来说，是比较简单的，Simulink3.0 以前的调试器都是基于命令行输入，而不是图形用户界面。所以在使用上有时可能不大方便。最新版的 Simulink 4.0，提供了一个图形化的调试界面，简化了调试操作，但是它在功能上和以前版本相比没有什么大的变化，只是为以前的一些命令提供了图形化的操作。

6.1　使用调试器

1. 启动调试器

使用 sldebug 命令或者 sim 命令的 debug 选项在调试器控制下启动一个模型。例如可以在 MATLAB 命令窗口输入：

> >sim ('vdp' ,[0,10],simset ('debut','on'));

或者

> >sldebut ' vdp';

这两个命令把 Simulink 的示例模型 vdp 载入内存并且在第一个仿真时间步暂停在执行顺序的第一个模块。调试器加亮显示模型的起始模块和模型图表中与之相关联的输出信号线。图 6－1 显示了 vdp 在调试器模式启动时的模块图表。

调试器还在 MATLAB 命令窗口显示仿真的开始时间和调试命令提示符。命令提示符显示模块的索引和将要被执行的第一个模块的名称。例如前面输入的命令会在命令窗口产生下面的输出：

```
[Tm = 0                    ]          * * Start * *  of system 'vdp' outputs
  (sldebug @ 0:0 'vdp/x1'):
```

这时，读者就可以通过在提示符后输入调试器命令或者其他的 MATLAB 命令进行操作，如获得调试器的帮助，单步运行仿真和检查数据等等。下面就来介绍如何使用这些调试器命令。

用户可以通过提示符后输入 help 命令来获得调试器命令的简短描述，而对于每一个命令的详尽描述可以查阅本章调试器命令参考一节。

Commands:

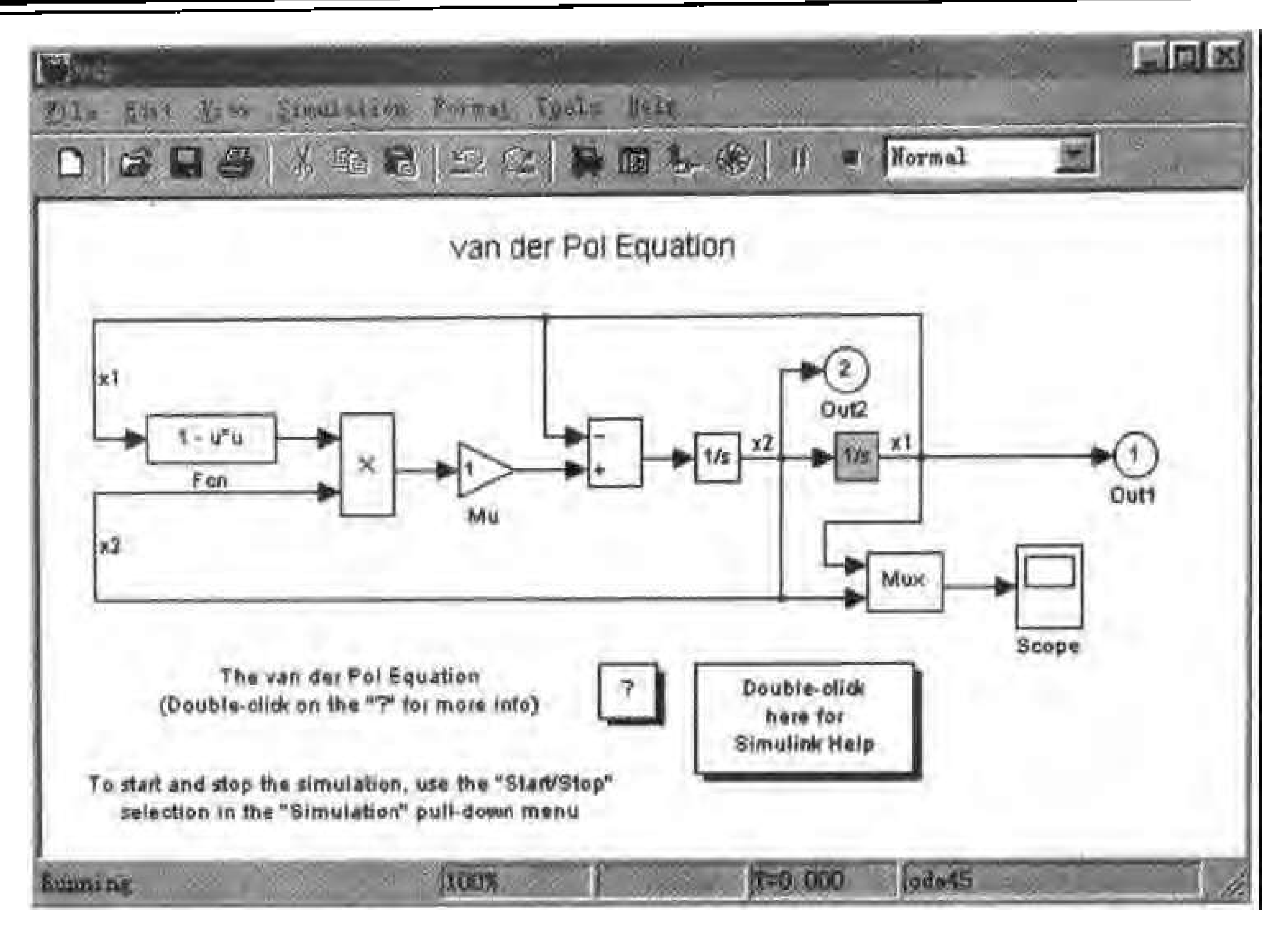


图 6-1 调试模式启动时的 vdp 模型

Step	Step to next block
Next	Go to start of next time step
Disp [s:b |gcb] Point	Register or display I/O of block (s) at every stopping
Undisp <s:b |gcb>	Remove a display point
Trace <s:b |gcb>	Add a trace point to display block I/O as it is executed
Untrace <s:b |gcb>	Remove a trace point
Porbe [s:b |gcb]	Probe I/O of block
Break <s:b |gcb>	Break before block is executed
Bafter <s:b |gcb>	Bradk after block is executed
Bshow s:b	Show block in system ,s, with sorted list index, b
Clear <s:b |gcb>.	Clear break point
Zcbreak	Toggle: break at monsampled zero crossing events?
Zclist	Display the nonsampled zero crossings list
Xbreak	Toggle: break when step size is limited by a state?
Tbreak[t]	Set/clear time break point
Nanbreak	Toggle: break on non-finite (NaN, Inf) values
Continue	Continue the simulation
Run	Stop debugging and finish the simulation
Stop	Stop execution

```
Quit              Abort the simulation
Status [all]      Display debugging actions in effect
States            Display current state values
Systems           Display a list of the model system
```

调试器允许用户输入调试命令的缩写形式，关于各个命令对应的缩写请读者朋友查看调试命令参考。Simulink 还允许用户空命令来重复前面执行过的命令。例如，读者输入 step 命令单步执行 vdp 模型，然后就可以不输入命令，直接按回车，调试器会重复前面的 step 命令，它如下所示：

```
(sldebug @0:0 'vdp/x1') :step
U1 = [0]
Y2 = [2]
(sldebug @0:1 'vdp/Outs'):
U1 =[2]
(sldebug @0:2 'vdp/Outs'):
U1 = [0]
Y1 = [0]
(sldebug @0:3 'vdp/Outs'):
U1 = [0]
```

2. 关于模块索引

Simulink 的许多命令和信息都是通过模块索引来指代模块。模块索引的通用形式为 s:b，其中 s 是一个标识当前模块在被调试模型中所处的系统的整数，而 b 则是标识该系统中的第几个模块。例如 0:1 表示模型中系统 0 的模块 1。Simulink 提供了一个 slist 命令来显示模块中各个模块的索引。

```
> > slist

- - Sorted list for 'vdp'[12 blocks, 9 nonvirtual blocks, directFeed = 0]
0:0    'vdp/x1'        (Integrator)
0:1    'vdp/Out1'      (Outport)
0:2    'vdp/x2'        (Integrator)
0:3    'vdp/Out2'      (Outport)
0:4    'vdp/Fcn'       (Fcn)
0:5    'vdp/Product'   (Product)
0:6    'vdp/Mu'        (Gain)
0:7    'vdp/Scope'     (Scope)
0:8    'vdp/Sum'       (Sum)
```

因为虚拟模块对模型没有实质性的作用，所以在调试时，Simulink 不会为这些模块指定索引。

3. 访问 MATLAB 工作空间

读者不但可以在调试命令提示符下输入调试命令，而且还可以输入任何有效的

MATLAB 命令。例如,读者可以在模型中设置一个断点,并把模型的时间和输出保存在 MATLAB 变量 tout 和 yout,然后就可以用下面的命令

(sldebug···): plot (tout, yout);

来绘制一个输出 - 时间曲线。

如果用户试图访问一个名称和 Simulink 调试器命令名称的部分或者全部相同的变量,例如是 s,它和 step 命令的一部分相同。如果用户在调试器提示符下直接输入 s,就意味着输入了 step。但是可以用下面的命令来显示变量 s 的值。

(sldebug···) : eval ('s')

6.2 增量运行模型

单步运行程序是几乎所有的调试器都具有的标准功能,Simulink 也不例外。Simulink 允许用户从一个模块跳转到另一个模块,从一个时间点到另一个时间点,从一个断点到另一个断点。用户可以选择表 6-1 中合适的仿真命令来推进仿真。

表 6-1 单步执行的调试命令

命 令	推进仿真的方式
Step	一个模块
Next	一个时间步
Continue	到下一个断点
Run	仿真的终点,并忽略断点

1. 按模块单步执行

若想让仿真按模块单步执行,可以在命令窗口输入 step 命令,该命令执行当前的模块后停止,并加亮显示模型中的模块执行次序中的下一个模块。例如,图 6-2 显示了执行完第一个模块后的模块图表。图 6-3 显示了执行完这个模块后命令窗口的情况。

如果紧接着要执行的模块出现在子系统里,那么调试器就会打开子系统的图表并加亮显示这个模块。在执行完一个模块后,调试器在命令窗口显示模块的输入和输出,并显示调试命令提示符,提示符里会显示下一个要执行的模块的索引。例如:

```
(sldebug @0:0 'vdp/x1') : step
U1 = [0]
Y1 = [2]
(sldebug @0:1 'vdp/Out1'):
```

2. 按时间单步执行

当用户执行完模型排序列表中的最后有一个模块时,调试器把模型推进到下一个仿真时间步,并暂停在下一个仿真时间步要执行的第一个模块的开始处。为了指示用户处于仿真时间步的边界,调试器在 MATLAB 命令窗口显示当前的时间。例如,执行完 vdp 第一个时间步的最后一个模块后,会产生如下的输出:

```
(sldebug @0:8 'vdp/Sum') :step
```

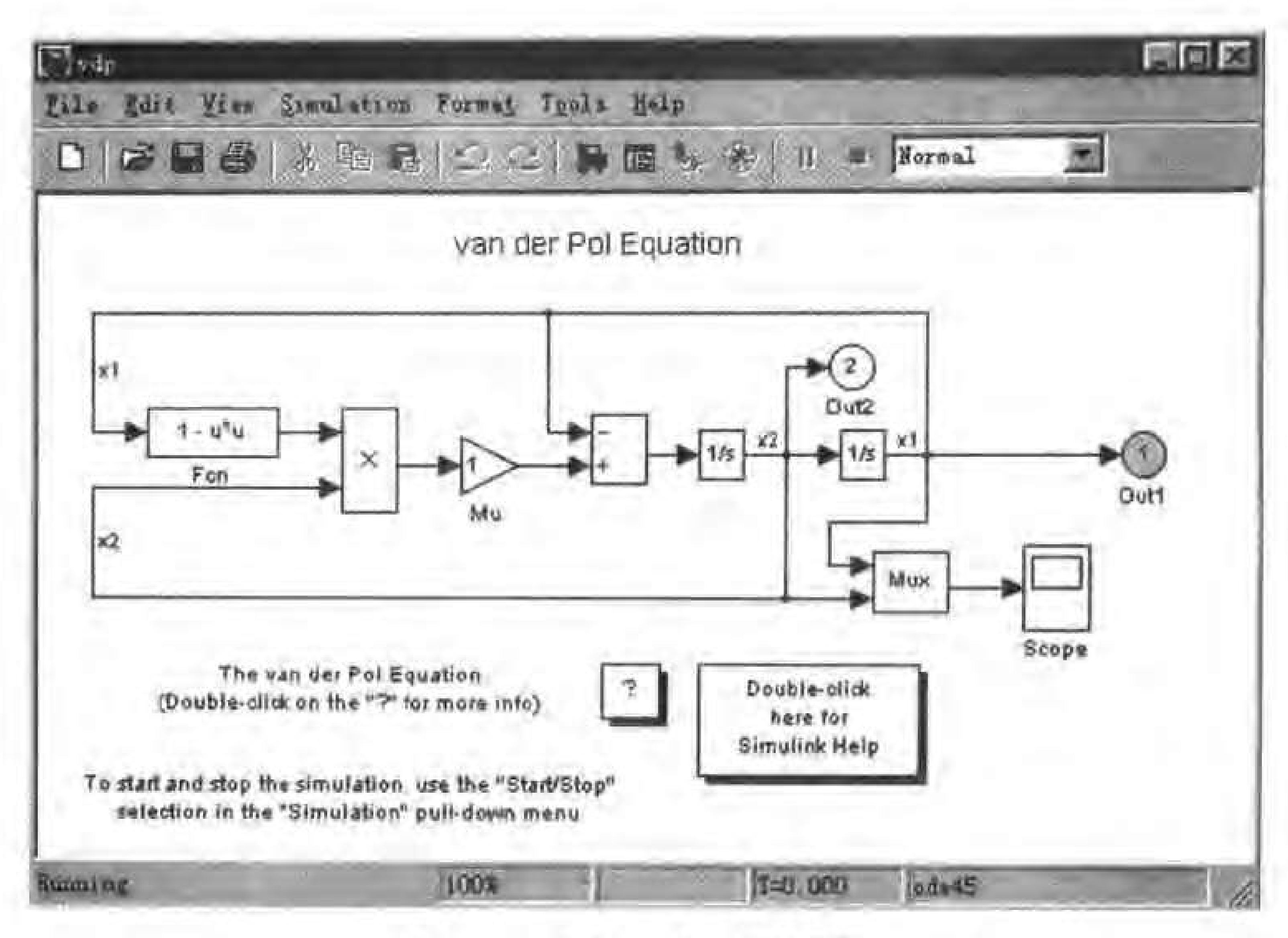


图 6－2　执行第一个模块后模型图表

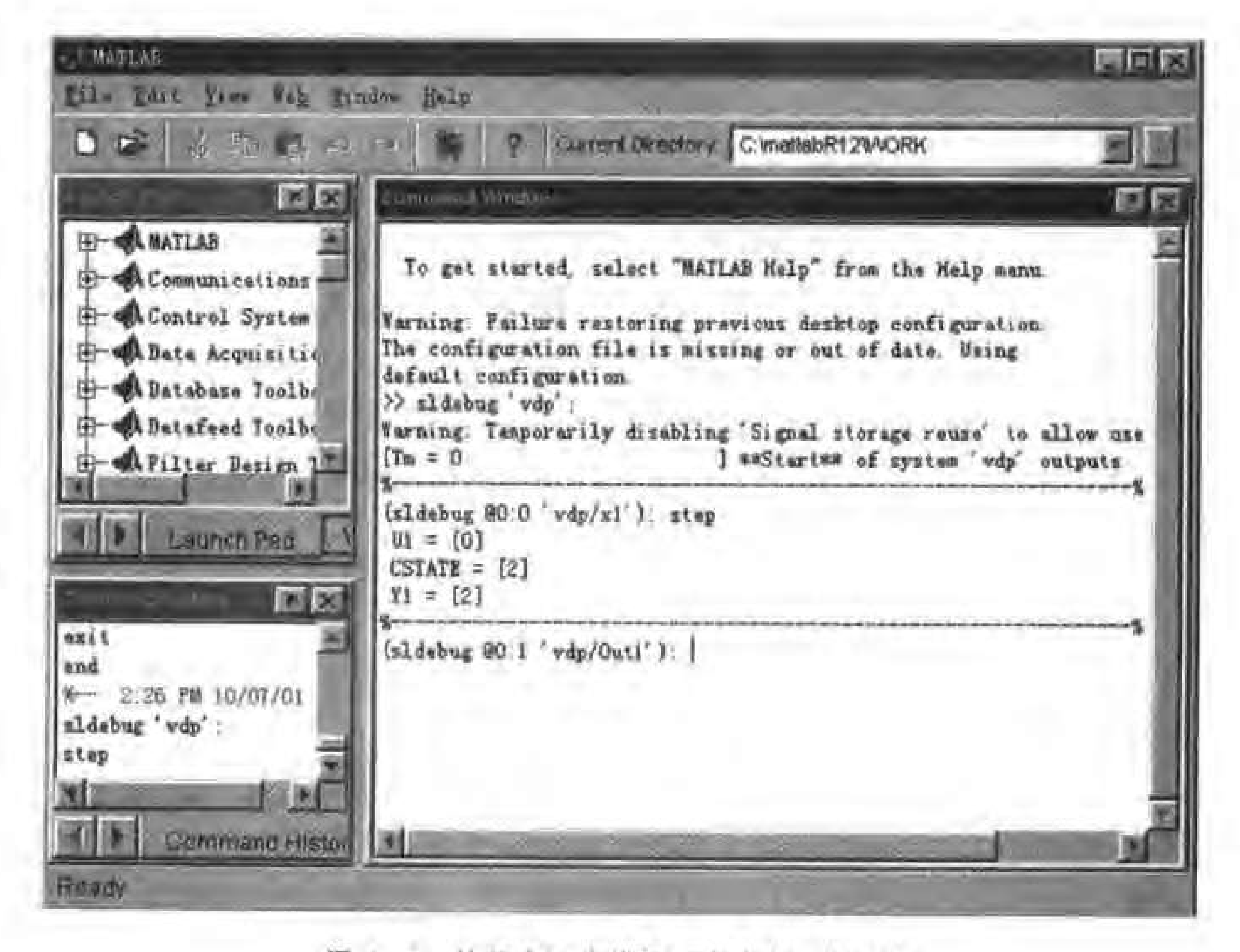


图 6－3　执行完这个模块后命令窗口的情况

```
U1 = [2]
U2 = [0]
Y1 = [-2]
[Time = 0.0001004754572603832      ] * * Start * * of system 'vdp' outputs
(sldebug @0:0 'vdp/x1'):
```

图 6-4 显示了 vdp 执行完一个时间步后命令窗口的情况。

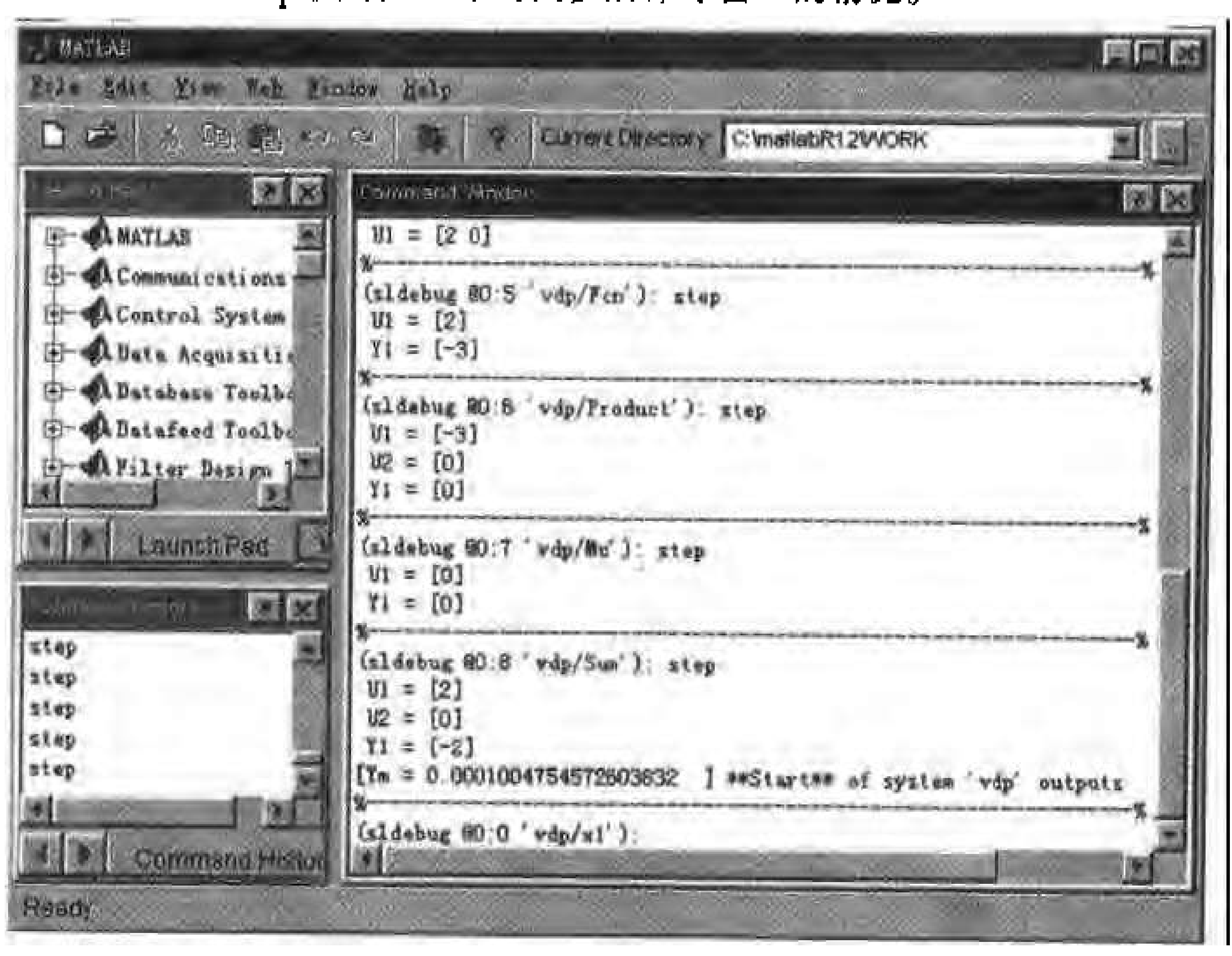


图 6-4 vdp 执行完一个时间步后命令窗口的情况

读者可以按最小的时间步单步执行仿真，也可以按最大的时间步来执行。这时，要使用命令 minor。

Next 命令执行当前仿真时间步中的其余模块。效果上，这个命令允许用户只通过一个命令直接跳到下一个仿真时间步，否则使用 step 则要一个模块接着一个模块执行，如果模型复杂就需要很多次命令了。Next 命令在用户对当前时间步剩余的模块的行为没有兴趣时使用就十分方便了。将仿真推进到下一个仿真时间步后，调试器停留在执行顺序表中的第一个模块。例如：

```
(sldebug @0:0 'vdp/x1') : next

[Tm = 0.0006028527435622993      ] * * Star * * of system 'vdp'outputs
(sldebug @0:0 'vdp/x1') :
```

3. 推进到下一个断点

continue 命令将仿真从当前的断点推进到下一个断点和仿真的终点两者中最先出现的一个。

如果用户确信整个仿真过程的余下部分没有感兴趣的事情发生，就可以使用 Run 模块从当前仿真点无中断地运行到仿真的末尾。注意，中间设置的任何断点对 Run 命令都不起作用。在仿真的终点，调试器将把控制权交给 MATLAB 命令行。这时如果用户想继续调试一个模型，就必须重新启动调试器，方法还是如前面讲到的。

6.3 设置断点

提供设置断点的能力也是调试器的标准功能之一，在前面一节讲到，continue 命令可以将仿真从一个断点运行到另一个断点，这样就提高了用户对仿真过程的控制能力。那么，在 Simulink 里用什么命令来设置断点呢？

Simulink 调试器允许用户设置的断点有两种类型：无条件的和条件的。无条件断点无论何种情况下，在仿真到达先前标记为断点的模块或者时间步时就生效，也就是使仿真暂停。而条件断点只有在用户预先定义的条件发生的情况下，才会生效。表 6－2 列出了可以用来设置断点的调试命令以及它们各自的作用。

表 6－2 用于设置断点的调试命令

命 令	产生的仿真中断点的位置
Break <gcb | s:b>	在模块索引 s:b 标识的模块的开始时
Bafter <gcb | s:b>	在模块索引 s:b 标识的模块的结束时
Tbreak [t]	在仿真时间步 t
Nanbreak	在上溢或者下溢(NaN)或者无穷大值(Inf)发生
Xbreak	当仿真到达决定仿真步长的状态
zcbreak	当过零点发生在仿真时间步之间时

6.3.1 非条件中断

1. 在模块处中断

Break 命令让用户在模块的开始处设置断点，这样，在每一个时间步，当仿真运行至这个模块都会停止。

指定模块的方法既可以是使用模块索引，也可以是使用图形方法。使用图形方法就要将 Simulink 的模型窗口和 MATLAB 命令窗口结合起来。方法是首先在模型图表里用鼠标选中要设置断点的模块（见图 6－5），然后在命令窗口调试命令提示符后输入下面的命令：

```
(sldebug @0:0 'vdp/x1'):break gcb %将断点设置在模型图中选中的模块的开始处
Break point 0:4 'vdp/fcn' installed.
(sldebug @0:0 'vdp/x1'):
```

图 6－6 显示了在命令窗口输入后的情况。

用户也可以用它的模块索引来标识模块。按下面的格式输入命令：

```
> > Break s:b
```

其中，s:b 是模块的索引，如果用户不清楚各个模块的索引，可以用 slist 来查询。例如 vdp 模块中的 Fcn 模块是 0:4。于是用下面命令也可以在模块的开始处设置断点。

```
> > Break 0:4
```

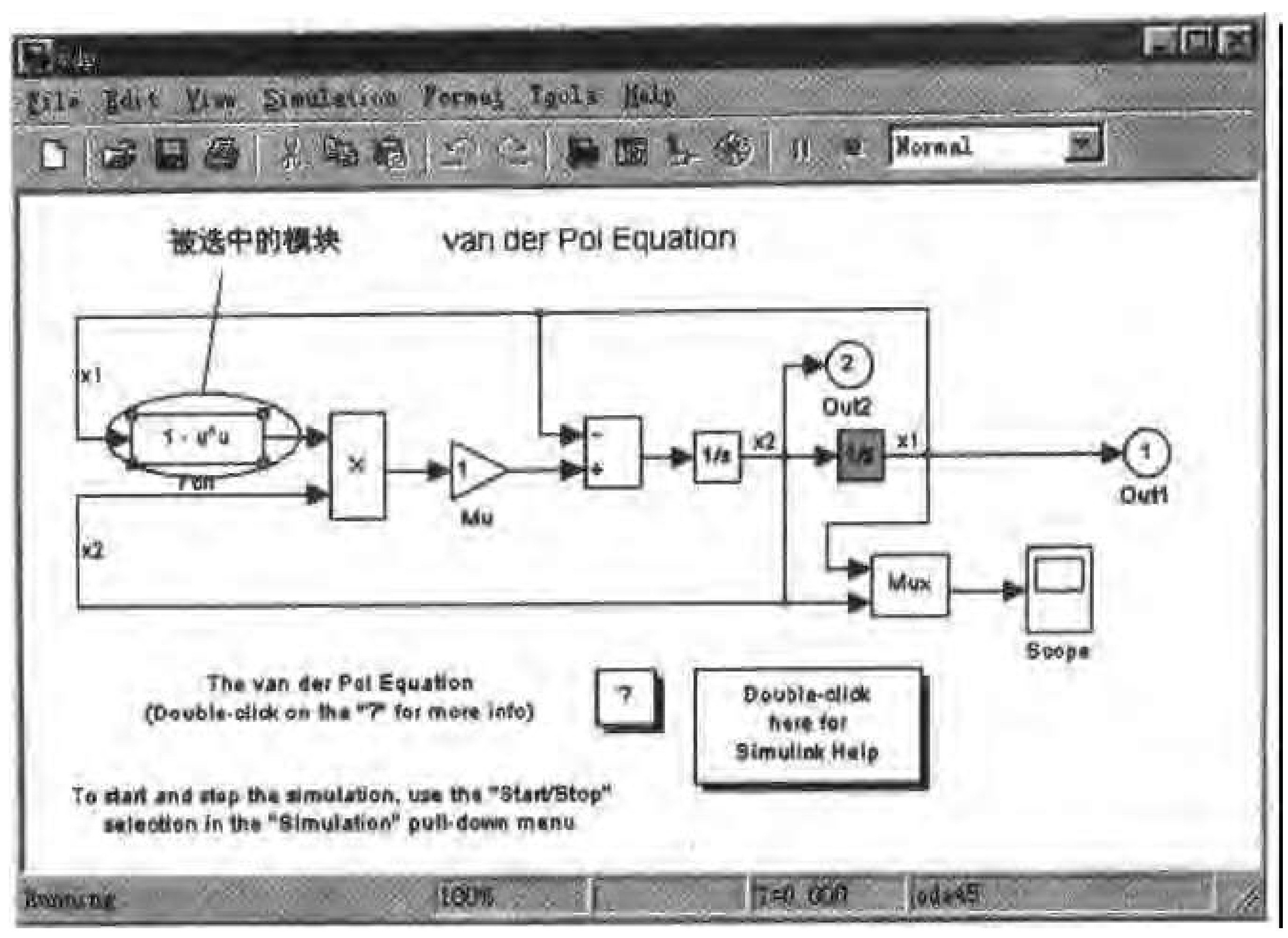


图 6－5　图形方法中选中模块示意

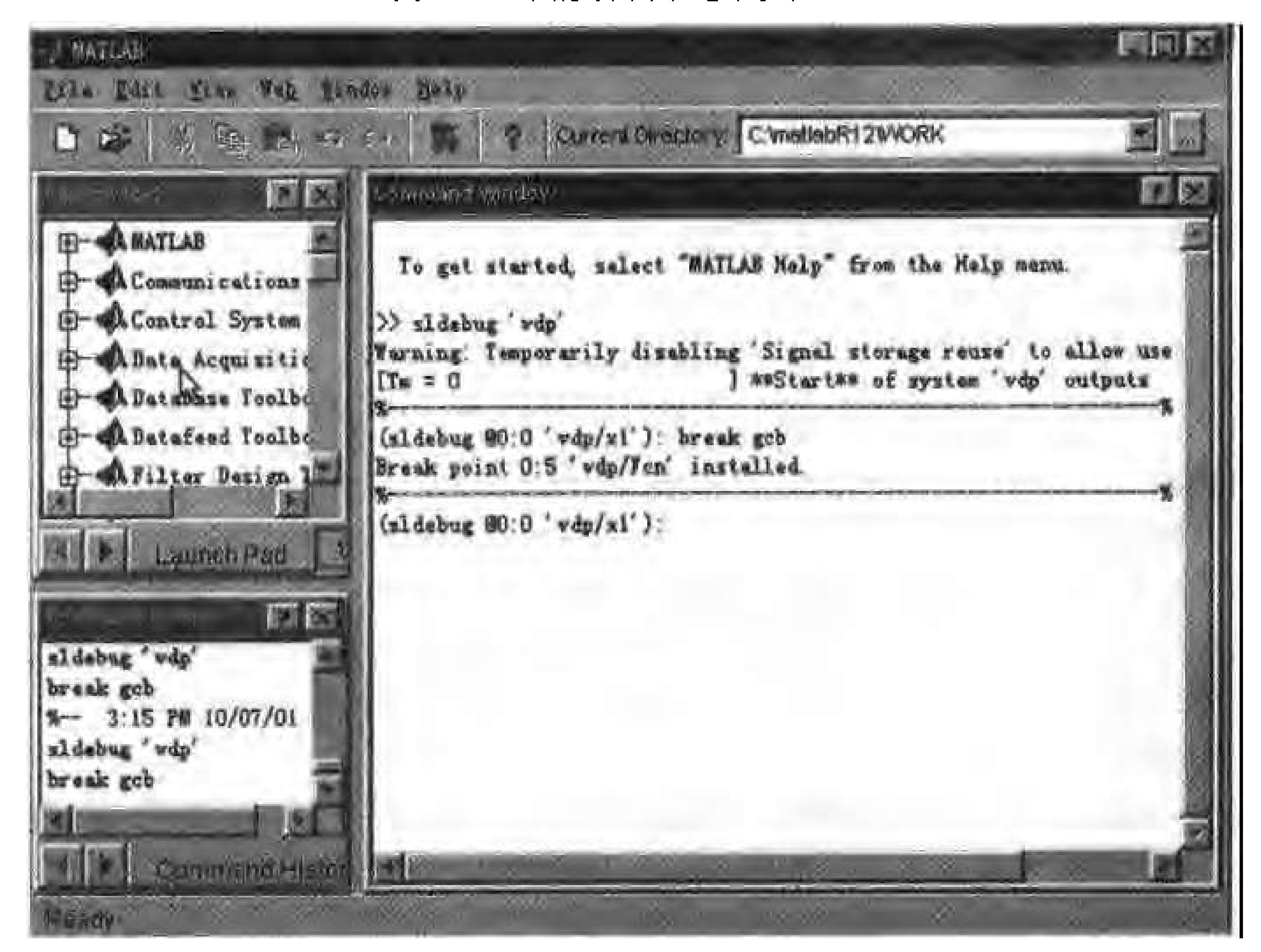


图 6－6　在命令窗口输入命令后的情况

使用类似的格式，用户可以用 bafter 命令在非虚拟模块的结束处设置断点。

清除断点的命令是 clear，它可以清除在模块中设置的断点。使用的方法也有模块索引和图形两种，这些和 Break 命令是很类似的。例如：

```
>>clear 0:4
```

或者，先在模型图表中选定 Fcn 模块，然后用 gcb 作为 clear 命令的输入参量。

```
>>clear gcb
```

注意，Simulink 调试器和许多高级语言清除断点的方法不一样，在那些高级语言里，在已经设为断点的地方再把它们设为断点，就会清除这些断点。但是在 Simulink 调试器里情况就不是这样。例如，前面已经设置过 0:4 模块的断点，如果再使用 Break 命令，并没有清除这个断点，调试器只是显示一个'vdp/Fcn'断点已经存在的信息。例如：

```
(sldebug @0:0 'vdp/x1'):Break 0:4
Break point @0:4 'vdp/Fcn' is already installed.
```

2. 在时间步中断

若想在特定的时间步设置断点，用户可以使用命令 tbreak。这个命令只需要输入一个时间参量 t 用来指定要设置断点的时刻，使用它之后，当仿真运行到 t 所指定的时刻时，仿真就中断。例如：

```
(sldebug @0:0 'vdp/x1') : tbreak5
(sldebug @0:0 'vdp/x1') : continue

Time Break point found (tbreak)
[Tm = 5.05795155935553          ] * * Start * * of system 'vdp'outputs
```

注意，参量 t 指定的时间是真正的时间，而不是仿真时间步的个数，如上例中就是指定在 5 s 处设置断点，但是调试器在运行时，却是在所有大于 t 的最接近的仿真时间步停止，所以仿真中断的时刻时 5.05…

6.3.2 条件中断

条件中断和非条件中断的区别是，条件中断下，模型被中断的具体时机的不确定性，即不改变中断设置，每次中断发生情况，也会不尽相同，它受到仿真运行具体情况的影响。而无条件断点则不同，例如，只要用户用 Break 在某个模块设好了断点，无论仿真的输入如何改变，只要仿真运行到这个模块，中断都会发生，即不受执行状况的影响。Simulink 用于设置条件中断的命令有：nanbreak，xbreak 和 zcbreak。

1. nanbreak

nanbreak 命令用来设置在仿真中出现无限大的值时发生的中断。设置了这种中断，如果在仿真中计算出无穷大的数或者是超出了运行仿真的机器的表示范围的数(上溢和下溢)，调试器将会中断仿真。nanbreak 在查找模型的计算性错误时，非常有用。

2. xbreak

用 xbreak 状态可以设置这样的中断，如果模型使用步长可变解法器，并且解法器遇到一个会限制解法器可以采用的步长的状态，调试器就中断仿真。这个命令在调试的模型出现需要更多的仿真步的情况时非常有用。

3. zcbreak

zcbreak 命令可以产生在出现过零点时发生的中断。这时，当 Simulink 检测到一个非采样过零点出现在模型里，或者是模型包括可能产生过零点的模块，调试器就暂停仿真。暂停之后，会在命令窗口显示中断在模型中的位置、时间和过零点的类型(rising 还是 failing)。例如，在 zerosxing 演示模型开始执行前设置一个过零点中断。

```
> > sldebug zeroxing            %在调试器模式下，执行 zeroxing 模型
```

[Tm = 0　　　　　　　　　　] * * Start * * of system 'zeroxing' outputs

Break at zero crossing events is enabled.

然后用 continue 命令继续仿真

(sldebug @0:0 'zeroxing/Sine Wave') : continue

[Tm = 0.34350110879329　　] Breaking at block 0:5

[Tm = 0.34350110879329　　] Rising zero crossing on 3rd zcsignal in block 0:5 'zeroxing/Saturation'

此时,zeroxing 的模型图表就加亮显示出现中断的模块,如图 6-7 所示。图 6-8 是命令窗口的情况。

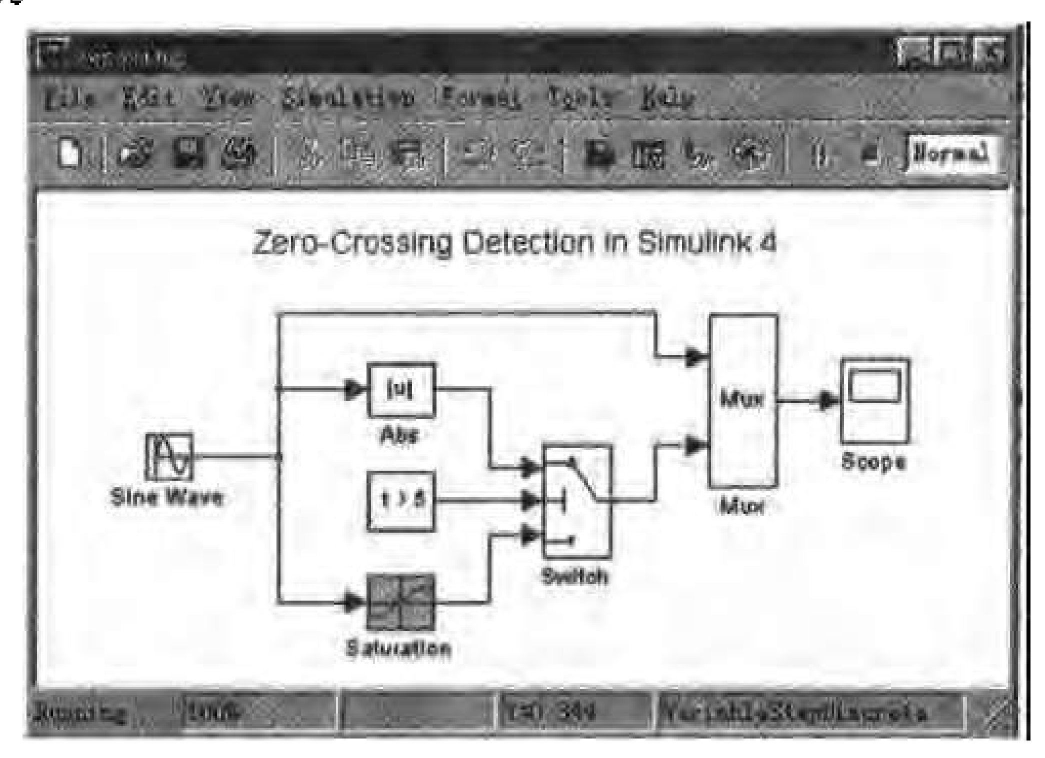


图 6-7　过零点中断示意

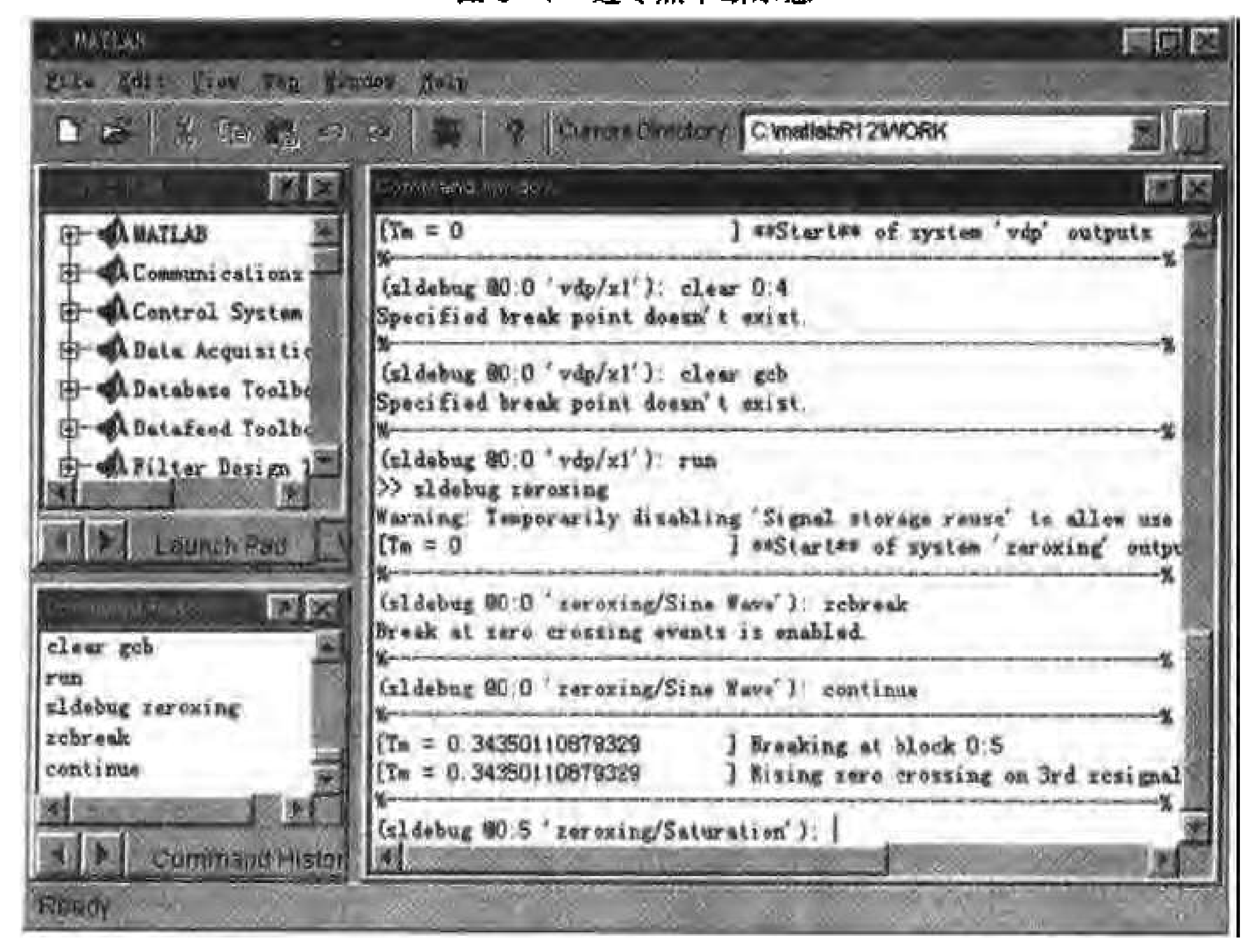


图 6-8　命令窗口的情况

如果用户为一个没有能产生非采样过零点的模块的模型设置过零点中断，调试器会显示信息来告诉用户这一点。

6.4 显示仿真有关的信息

Simulink 调试器提供了许多命令允许用户在模型运行时，显示模块的状态、模块的输入输出和其他的信息。

6.4.1 显示模块的输入和输出(I/O)

调试器提供了三个命令来显示模块的 I/O，这些命令都可以显示指定模块的输入和输出，它们之间的不同之处在于什么时候显示 I/O。这三个命令分别是：probe、disp 和 trace。

1.probe

probe 命令显示用户通过命令参量指定的模块的输入和输出，显示的位置是在 MATLAB 命令窗口。表 6-3 列出了 probe 命令的不同用法。

表 6-3 probe 命令的不同用法

命 令	描 述
probe	进入或者退出 probe 模式，在 probe 模式下，调试器显示用户在模型图表中所选择的任何模块的当前输入和输出。这时，在提示符后输入任何命令都会使调试器退出 probe 模式
probe.gcb	显示被选中模块的输入和输出
probe s:b	显示由索引 s:b 所指模块的输入和输出

probe 命令可以方便地用于显示非当前执行模块的输入和输出。因为在调试时，比如，用 step 命令，调试器只会自动地显示最近执行完的模块的输入和输出，而 probe 命令可以让用户查询其他模块现有(最近时刻)的输入和输出。而 next 命令直接跳到下一个仿真时间步，执行过程中，调试器不显示模块的输入和输出，这时，就可以使用 probe 命令来检查模块的输入和输出。例如：

```
(sldebug @0:5 'vdp/Product') : step
%仿真已经运行到模块 0:5，调试器显示了它的输入和输出
U1 = [-3]
U2 = [0]
Y1 = [0]
(sldebug @0:6 'fcp/Mu') : probe 0:2
%此时，想查询前面执行过的模块 0:2 的输入和输出
I/O of 0:2 'vdp/x2':
U1 = [0]
Y1 = [0]
(sldebug @0:6 'vdp/Mu') : probe 0:6
```

%也可以查询在当前仿真时间步尚未执行的模块 0:6 的输入和输出,这时显示的是该模块在前一个仿真步的输入和输出,如果当前处于第一个仿真步,那显示的就是它的初始的输入和输出。

I/O of 0:6 'vdp/Mu':

U1 = [0]

Y1 = [0]

(sldebug @0:6 'vdp/Mu') :step

%模块 0:6 执行后,它的输入和输出。

U1 = [0]

Y1 = [0]

(sldebug @0:7 'vdp/Scope') : next

%使用 next 命令将模型推进到下一个仿真时间步,显示器没有显示任何模块的输入和输出,而只是显示 next 命令执行后仿真处于的时间。

[Tm = 0.0001004754726O3832] * * Start * * of system 'vdp'outputs

(sldebug @0:0 'vdp/x1') : probe 0:7

%于是,用 probe 0:7 命令来查询模块 0:7(scope)的输入和输出。

I/O of 0:7 'vdp/Scope':

U1 = [1.999999989905697 -0.0002009206312731412]

(sldebug @0:0 'vdp/x1'): probe 0:8

%查询模块 0:8 的输入和输出,用户可以用 probe 进入 probe 模式,然后在模型图表中选择 Sum 模块,调试器会自动在命令窗口显示 Sum 模块的输入和输出。

I/O of 0:8 'vdp/Sum' :

U1 = [1.999999989905697]

U2 = [0.0006027618857068085]

Y1 = [-1.99939722801999]

2.disp

disp 命令的作用是使调试器在仿真暂停的任何时候显示 disp 命令参数指定的模块的输入和输出。指定模块的方式既可以通过模块索引,也可以通过图形方式——先在模型图表中选择模块,然后输入 gcb 作为 disp 命令的输入参量。用户可以用 undisp 命令将一个模块从调试器的显示模块列表中去掉,它同样支持模块索引和图形两种方式。

disp 命令是用来跟踪特定的一个或多个模块在用户单步执行仿真时的输入和输出。使用 disp 命令指定这些模块后,调试器会在每一次单步执行停止时显示这些模型的输入和输出。但是要注意,由于使用 step 命令时,调试器会自动地显示当前执行模块的输入和输出,如果用户仅仅需要这样,就不必使用 disp 命令了。例如,紧接着前面的 vdp 模型的执行,用户需跟踪模块 Fcn 的输入和输出。

(sldebug @0:0 'vdp/x1') : disp 0:5

%把 0:5 模块加入到跟踪列表里

Display of block 0:5 'vdp/Product' installed.

(sldebug @0:0 'vdp/x1') : next

%使用 next 命令,按理是不显示任何模块的输入和输出的,但是因为 0:5 模块需要被跟踪,所以调试器显示了它的输入和输出。

```
[Tm = 0.0156741713321978      ] I/O of block 0:5 'vdp/Product'
U1 = [-2.999032582157]
U2 = [-0.03062158372261323]
Y1 = [0.0918351272998699]
[Tm = 0.015674717133261978 ] * * Start * * of syastem 'vdp' outputs

(sldebug @0:0 'vdp/x1')
[Tm = 0.07847133212035928 ]I/O of block 0:5 'vdp/Product'
U1 = [-2.977233198865903]
U2 = [-0.1397561358249672]
Y1 = [0.4160866073233047]
[Tm = 0.07847133212035928 ] * * Start * * of system 'vdp'outputs
(sldebug @0:0 'vdp/x1') : udisp 0:5
%将模块 0:5 从显示模块列表中去掉
```

注意,如果使用 step 命令单步执行,使用 disp 命令的效果并不明显。

3. trace

trace 命令作用是使调试器显示指定的模块的输入和输出。使用了这个命令的模块,Simulink 无论何时对它们进行估值,调试器都会显示它们的输入和输出。它让用户不中断仿真就可以获得一个模块输入和输出的完整记录。与别的显示信息命令一样,trace 同样支持模块索引作为参量和 gcb 作为参量的图形方式这两种格式。如果要把一个模块从调试器的 trace 列表中去掉,可以使用 untrace 命令,它同样支持前面常提到的两种格式。例如:

```
[Tm = 0.07847133212035928 ] * * Start * * of system 'vdp'outputs
(sldebug @0:0 'vdp/x1') : tbreak 1
%在 1 s 处设置一个断点
(sldebug @0:0 'vdp/x1') : continue
%将仿真停止在前面所设置的断点前
.......
[Tm = 1.2788459829406        ] * * Start * * of system 'vdp'outputs
(sldebug @0:0 'vdp/x1') : trace 0:4
%将模块 0:4 加入到调试器的 trace 列表中
Trace of block 0:4 'vdp/Fcn' installed.
(sldebug @0:0 'vdp/x1') : tbreak 1.5
%在 1.5 s 处设置断点
(sldebug @0:0 'vdp/x1') : continue
%仿真将运行至约 1.5 s 处停止,这期间模块 0:4 的输入和输出将被显示。
[Tm = 1.27878459829406       ] I/O of block 0:4 'vdp/Fcn'
```

```
U1 = [1.268741682038]
Y1 = [ - 0.60970544250098887]
Time break point found (tbreak). %在时间断点处停止
[Tm = 1.67878459829406      ] I/O of block 0:5 'vdp/Product'
U1 = [0.32230544328271l3]
U2 = [ - 1.319256553049167]
Y1 = [ - 0.4252035681341334]
[Tm = 1.67878459829406      ] * * Start * * of system 'vdp' outputs
```

从上面的例子可以看到,在模型运行中(从一个断点运行到另一个时间断点),仿真不用停止就可以显示模块 0:4 在各个仿真时间步的输入和输出,至于其中用了两个中断,则是为了使要显示执行的结果不要太长,方便举例。

6.4.2　显示代数环信息

atrace 命令可以使调试器显示模型中的代数环在每个时间步内被求解的有关信息。atrace 命令只有一个参量,它的用法如表 6 - 4 所示。

表 6 - 4　atrace 命令的使用格式

命　令	显示的信息
atrace 0	没有信息显示
atrace 1	环路变量的解,求解环路所需的迭代次数,以及估计的解的误差
atrace 2	同 actrace 1
atrace 3	actrace 2 所显示的信息加上求解环路的雅可比(Jacobian)矩阵
atrace 4	actrace 3 所显示的信息加上环路变量在每次迭代的瞬时解

6.4.3　显示系统

states 调试命令列出在 MATLAB 命令窗口显示系统状态的当前值。例如,下面的命令序列显示了 Simulink 的反弹球示例模型(bounce)在第一、第二个仿真时间步的状态值。

```
Sldebug bounce
[Tm = 0               ] * * Start * * of system 'bounce'outputs
(sldebug @0:0 'bounce/Position') : state
%显示状态
Continuous state vector (value, index, name) ;
10                    0       (0:0 'bounce/Position')
15                    1       (0:5 'bounce/Velocity')
(sldebug @0:0 'bounce/Position') : next
[Tm = 0.01                     ] * * Start * * of system 'bounce' outputs
(sldebug @0:0 'bounce/Position') : states
```

```
%状态值的显示格式
Continuous state vector (value, index, name) :
10:1495095                 0  (0:0 'bounce/Position')
14.9019                    1  (0:5 'bounce/Velocity')
```

6.4.4 显示积分信息

ishow 命令将锁牢积分信息的显示。当它被使能时,这个选项使调试器显示它每进行一时间步或者遇上约束仿真时间步大小的状态时的时间信息。对应前面一种情况,调试器显示仿真时间步的大小,例如:

```
(sldebug @ 0:1 'bounce/Bouncing Ball Display') : next
% 对于第一种情况
[Tm = 0.03            ] Step of 0.01 was taken by integrator
[Tm = 0.03            ] * * Start * * of system 'bounce'outputs
```

%对于第二种情况,调试器显示限制仿真时间步大小的状态名和具有该状态的模块名。

```
(sldebug @0:0 'bounce/Position') : ishow
Display of integration information is enabled.
(sldebug @0:0'bounce/Position') : next
[Ts = 0.02            ] Integration limited by 1si state of block 0:0 'bounce/Posi-
                        tion'
[Tm = 0.02            ] Step of 0.01 was taken by integrator
[Tm = 0.02            ] * * Start * * of system 'bounce'outputs
```

6.5 显示模型的信息

调试器除了能显示关于仿真的信息外,还可以显示关于模型的信息。它们包括:显示模型中模块的执行次序表,显示模块,显示模型中的非虚拟系统和非虚拟模块,显示潜在的具有过零点的模块,显示代数环以及显示调试器的设置。下面就对它们一一进行介绍。

1. 显示模型中模块的执行顺序列表

我们知道 Simulink 要在模型运行开始时,模型的初始化期间,确定模块执行的次序列表。在仿真期间,Simulink 保存这个按执行顺序排列的列表。这里不妨称为排序表。用户可以在任何时候在调试命令提示符后输入 slist 命令来显示排序表,显示的列表包括了各个模块的索引。实际上,模块索引就是按执行次序来排列的。例如,显示 vdp 模型的排列表。

```
(sldebug @0:0 'vdp/x1') : slist
- - - - Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks, directFeed = 0]
0:0  'vdp/x1' (Integrator)
0:1  'vdp/Out1' (Outport)
0:2  'vdp/x2' (Integrator)
```

0:3 'vdp/Out2' (Outport)

0:4 'vdp/Fcn' (Fcn)

0:5 'vdp/Product' (Product)

0:6 'vdp//Mu' (Gain)

0:7 'vdp/Scope/ (Scope)

0:8 'vdp/Sum' (Sum)

由于只有非虚拟模块才会按执行次序排列,所以slist命令的另一个作用就是显示模型中的非虚拟模块。

2. 显示模型中的非虚拟系统

system命令可以显示模型中所有非虚拟系统的一个列表。读者可能对非虚拟系统这个概念有疑问。粗略地讲,非虚拟系统和非虚拟模块的意义是差不多的。所谓虚拟系统是指那些仅仅起到图形组织功能的系统,即这样的一些子系统,它们的模型图表尽管表示为子系统模块,但是在仿真时,Simulink把它内部包含的模块作为父系统的一部分。在Simulink模型里,只有根系统和条件子系统(触发或者使能子系统)才是真正的系统——非虚拟系统,而其他的所有子系统都是虚拟子系统,都不会出现在system命令显示的系统列表里。例如Simulink的cluth示例模型里有许多的子系统,但system只显示了根系统和两个触发子系统。

```
> >sldebug cluth          %在命令窗口输入
[Tm = 0                    ] * *Start* * of system 'clutch' outputs
(sldebug @0:0 'clutch/Clutch Pedal') : systems

0        'clutch'
1        'clutch/Locked'
2        'clutch/Unlocked'
```

注意,对应已经封装后的子系统,Simulink将它们作为模块来处理,同样不作为非虚拟系统来显示。

3. 显示具有过零点的潜在模块

zclist命令可以列出模型中所有可能会产生非采样过零点的模块的列表。例如,zclist命令列出clutch模块的所有产生过零点的潜在模块。

```
(sldebug @0:0 'clutch/Clutch Pedal') : zclist

2:3     'clutch/Unlocked/slip direction' (Signum)
0:4     'clutch/Friction Mode Logic/Lockup Detection/Velocities Match' (HiCross)
0:10    'clutch/Friction Mode Logic/Lockup Detection/Required Friction for Lockup/
        Abs' (Abs)
0:12    'clutch/Friction Mode Logic/Lockup Detection/Required Friction for Lockup/
        Relational Operator' (RelationalOperator)
0:19    'clutch/Friction Mode Logic/Break Apart Detection/Abs' (Abs)
0:20    'clutch/Friction Mode Logic/Break Apart Detection/Relational Operator' (Re-
```

```
        lationalOperator)
0:24    'clutch/Unlocked' (SubSystem)
0:27    'clutch/Locked' (SubSystem)
```

4. 显示代数环

ashow 命令加亮显示一个特定的代数环或者包含某个特定模块的代数环。加亮显示一个特定代数环的方法是 ashow s# n,其中,s 指包含该代数环的系统的索引,n 指代数环在该系统内的索引。如果要加亮显示包含特定模块的代数环,可以使用 ashow gcb,这个命令将会加亮显示用户在模型图表中当前选中的模块(即图形方式)。当然,用户也可以通过 s:b 这种模块索引格式来指定模块。如果要清除代数环的加亮显示,可以使用 ashow clear 命令。

5. 显示模块设置

status 命令用来显示模型中不同调试选项的设置情况,比如是否设置了条件断点等。下面的命令序列显示了 vdp 模型的初始调试设置。

```
Sldebug vdp
[Tm = 0                          ] * * Start * *  of system 'vdp'outputs
(sldebug @0:0 'vdp/x1') : status
Current simulation time                         :0 (MajorTimeStep)
Default command to execute on return/enter      : "  "
Stop in minor times steps                       : disabled
Break at zero crossing events                   : disabled
Break when step size is limiting by a state     : disabled
Time break point                                : disabled
Break on non - finite (NaN.Inf) value           : disabled
Number of installed break points                : 0
Number of installed display points              : 0
Number of installed trace points                : 0
Display of integration information              : disabled
Algebraic loop tracing level                    : 0
```

读者可以试着设置一个时间断点,再使用 status 命令显示调试器设置,会发现 Time break point 这一项的值变成了 enabled。

6.6 Simulink4.0 的图形调试工具

在 Simulink3.0 以前的版本中,对模型进行调试都是使用命令行命令,经常使用 VC ++ 等高级语言的读者可能觉得很不习惯,事实上也的确不是很方便。在 MathsWork 公司最新推出的 Simulink4.0 里,增加了个图形调试工具——Simulink Debugger。这一节就对它进行简单的介绍,详细的情况,请读者参考它的帮助文件。

读者可以通过 Tools 菜单下的 Simulink Debugger 命令来打开调试器,也可以通过工具栏上的快捷按钮(爬虫状,可以将鼠标移到按钮上,在 Simulink 模型窗口底部的状态栏

可以看到当前按钮的说明信息)来打开。图 6-9 就是对 zeroxing 演示模型进行调试时，调试器打开后的样子。

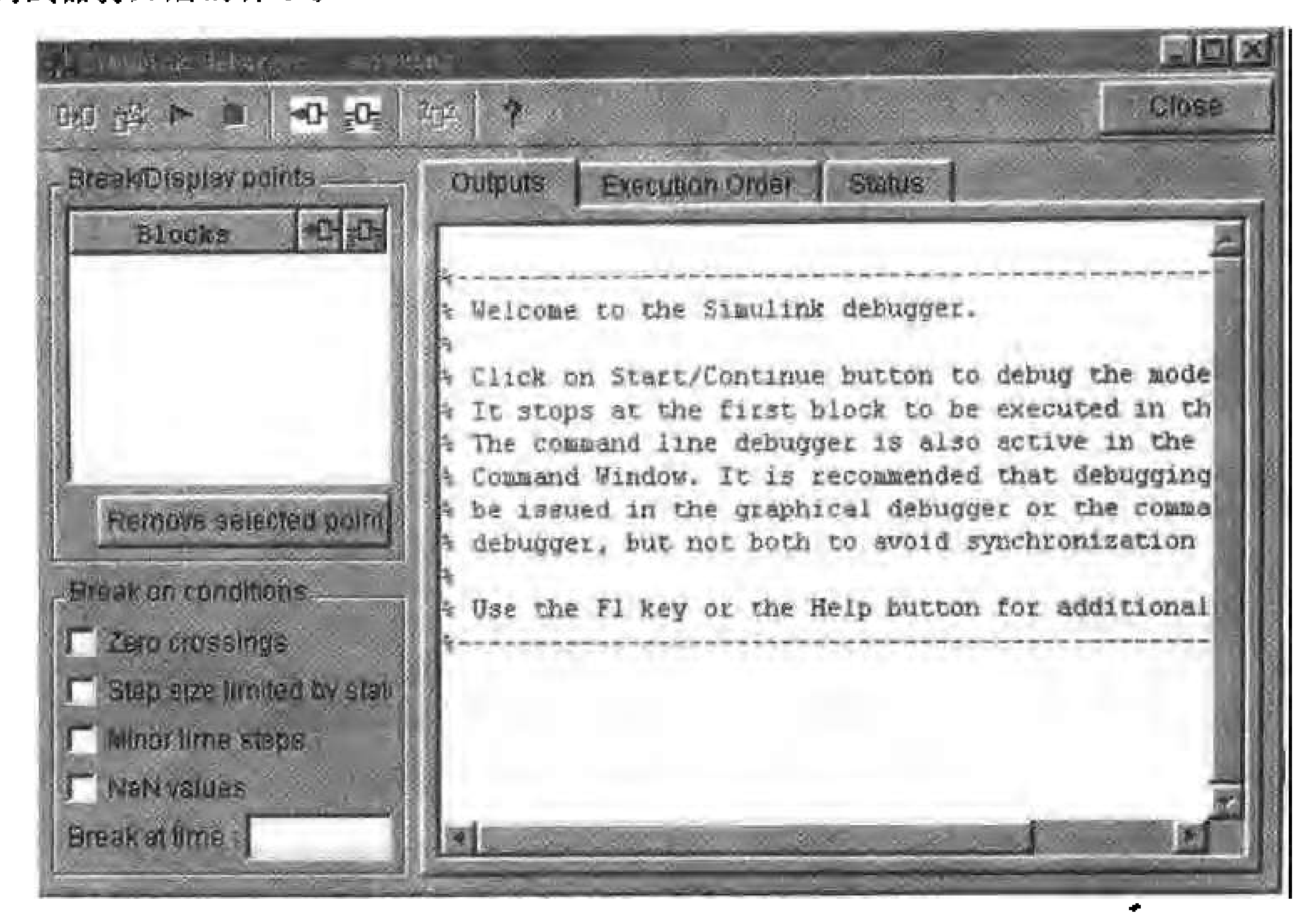


图 6-9 Simulink Debugger

整个调试器窗口分为两个部分，右边的窗口用于显示模型的信息，它分为三个页面：Outputs，Execution Order 和 Status。在调试器刚被打开时，这些页面什么也不显示，因为仿真的调试还没有开始。开始调试，可以按工具栏上的快捷按钮(向右的箭头)。

一旦开始了调试，右边的信息显示窗口就会显示各自的信息。例如，Execution Order 页面就显示 vdp 模型中各个模块的更新次序(排序表)，同时也给出了各个模块的模块索引，或者说是模型中的非虚拟模块列表，这个信息在模型调试中不会变化。而 Status 页显示的信息是指调试器的当前设置，它的作用和在命令行输入 Status 命令是一样的。这一页显示的信息会随着用户的操作而变化，例如用户设置新的断点，都会对它有影响。

而 Outputs 页则用于调试命令执行后，输出结果的显示，它和使用命令行命令调试时，MATLAB 命令窗口的作用一样。图 6-10 是 zeroxing 模型在开始调试时，调试器的 Outputs 页的输出。图 6-11 是 Execution Order 页的输出。图 6-12 是 Status 页的输出。

调试窗口的左边部分和工具栏的快捷按钮则是一些调试操作的图形命令。工具栏上的按钮的功能描述如下，从左往右依次是：

(1)运行仿真到下一个模块。

(2)运行仿真到下一个时间步的开始。

(3)开始或者继续运行仿真。

(4)终止模型的调试。

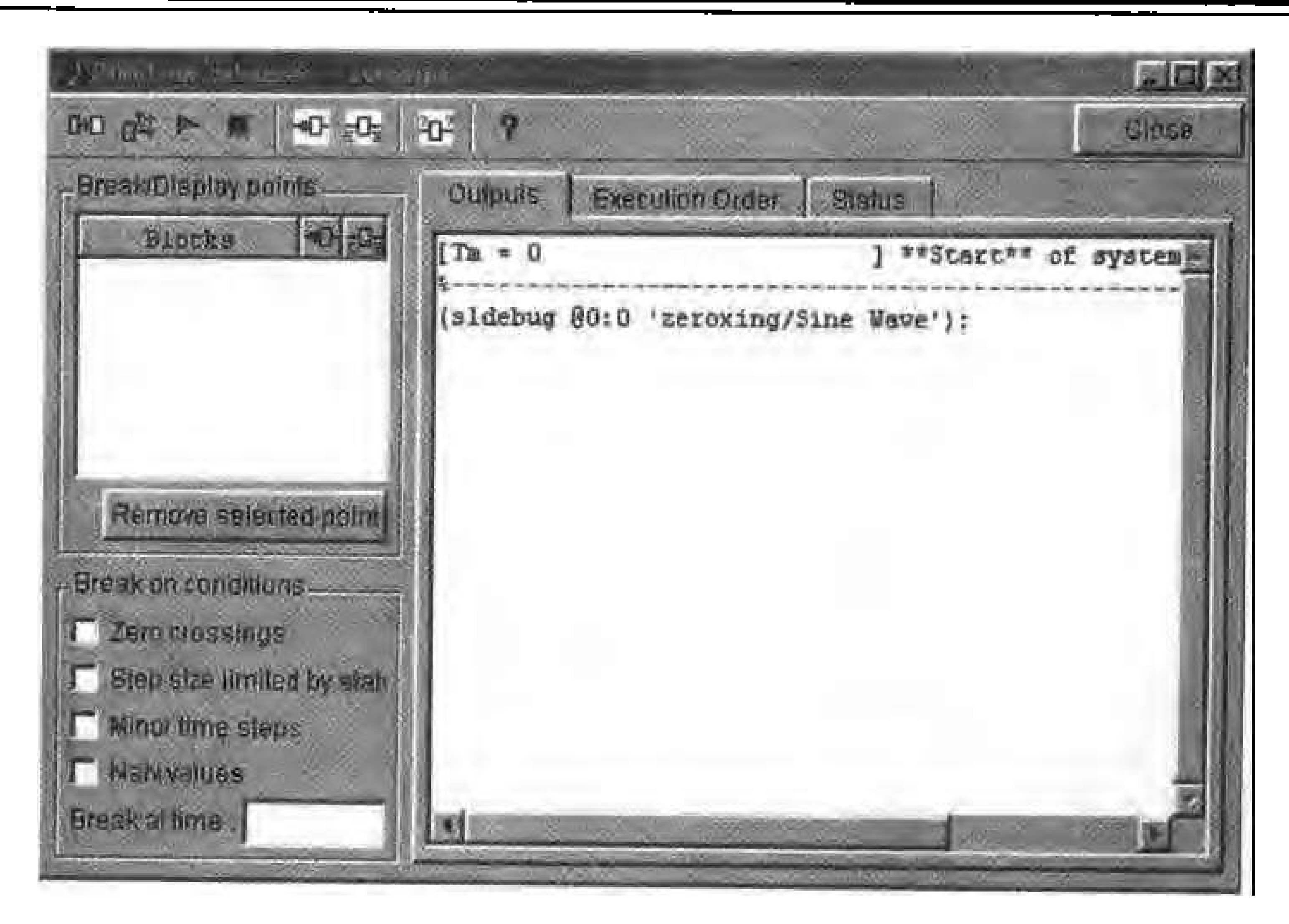


图 6-10 zeroxing 模型在开始调试的情况

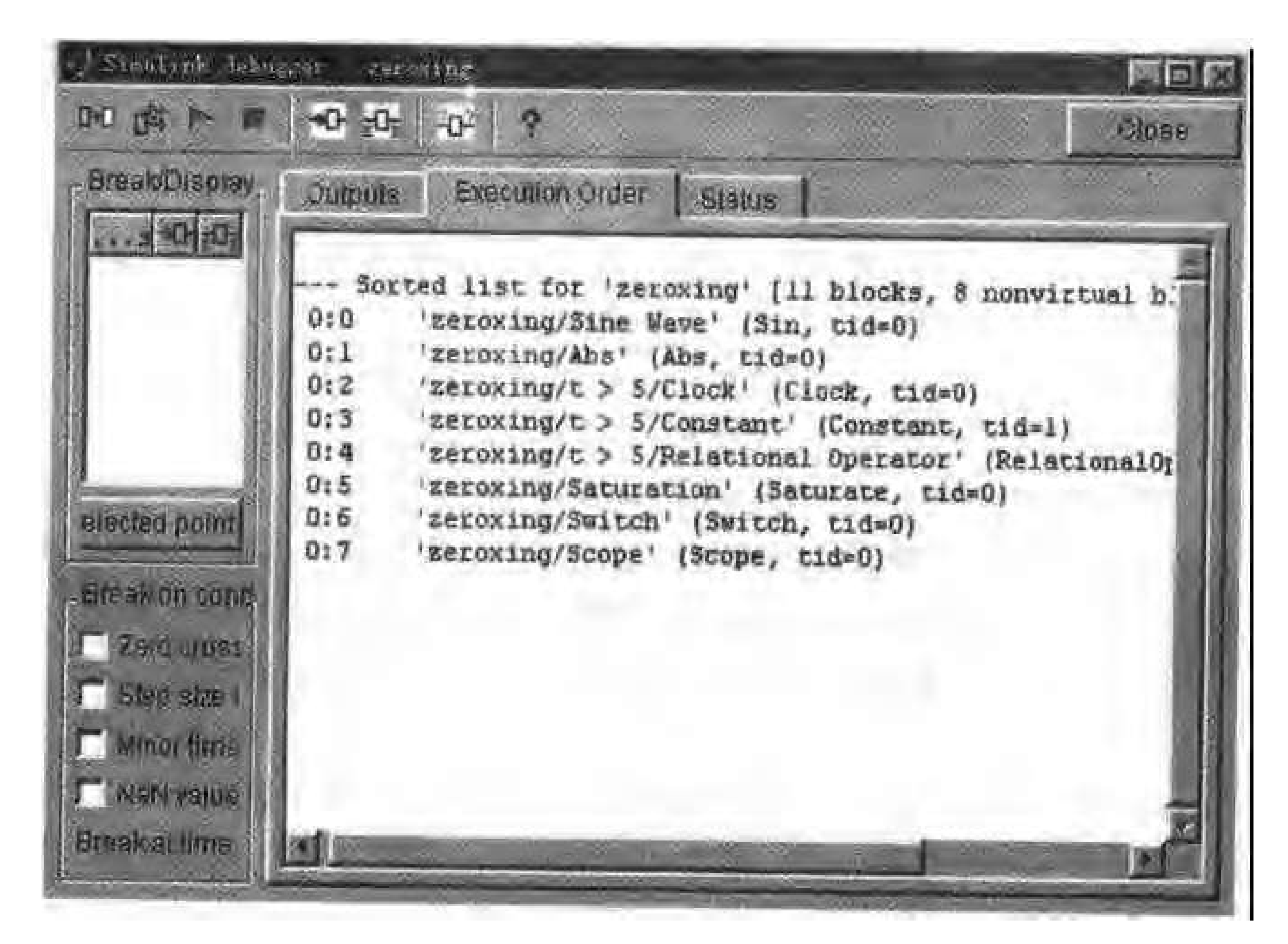


图 6-11 Execution Order 页的输出

(5)在当前所选择模块运行前设置断点。

(6)在所选择模块被执行时,显示它的输入和输出,即设置显示点。

(7)显示所选择模块当前仿真时间步的输入和输出。

关于这些操作的意义,请读者查看本章的前面几节。

这里要请读者注意,无论是设置断点,还是设置显示点,操作时都要回到模型窗口,先

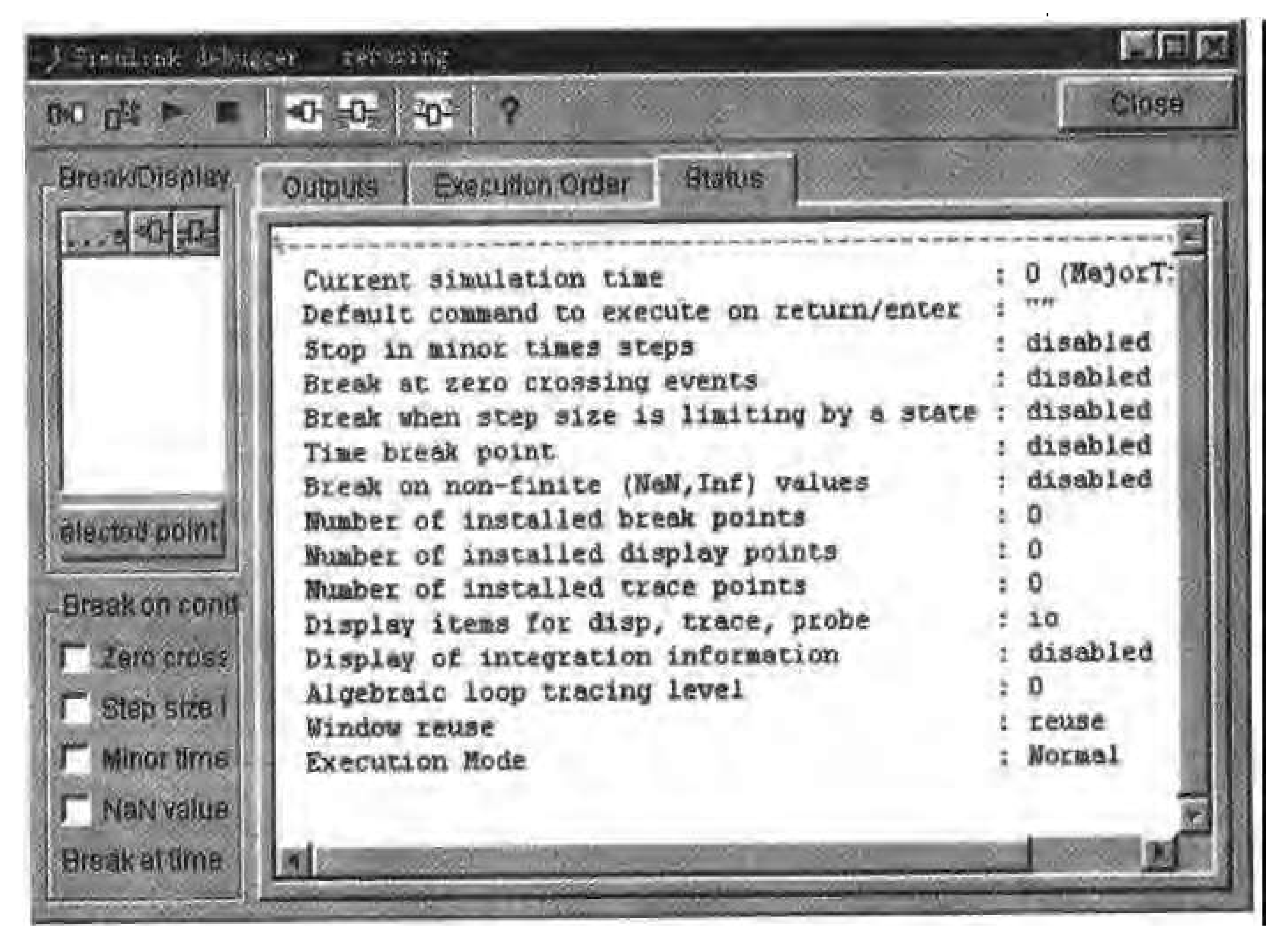


图 6-12　Status 页的输出

选中某个模块,然后再进行相应的操作,也就是采用前面所说的图形操作方式。

对于设置了断点或者是显示点的的模块,调试器会自动的把它们的模块名称添加到 Breaks/Display points 列表框。

Simulink 调试器还在窗口的左下角提供了几个检查框选项让用户进行条件中断点的设置,它们是:

(1)Zero crossings, 选中它表示遇到过零点检测时产生中断点;

(2)Step size limited by state, 选中它表示在步长受状态限制时产生中断点;

(3)Minor steps, 选中它,则使仿真进入最小时间步模式;

(4)NaN values, 选中它表示如果在仿真时一个计算值为无限大或者超出了机器表示范围时产生中断点。

关于这些设置的详细描述也请读者查看前面的章节。

调试窗口左边最下端的 Break at times 编辑框的作用是让用户设置在某个时间步设置断点,用户只须在里面输入要设置断点的实际时间,注意不是仿真的步数。

6.7　调试命令使用详解

为了方便读者查询 Simulink 的各种调试命令,在本章的最后给出这些命令的使用参考,表 6-5 列出了这些命令。其中,是否可重复这一列,表示改命令是否可以通过用回车键(return)来重复执行。例如,在调试命令提示符后输入 step 命令后,就可以通过按回车键来重复执行 step 命令,这省去了不少命令输入的工作。在表 6-5 中就详细地列出了每个命令的目的、语法、参量和说明信息。

表 6－5　调试命令集

命　令	简写形式	是否可重复	描　　述
Ashow	As	否	显示代数环
Atrace	At	否	设置代数环的跟踪等级
Bafter	Ba	否	插入一个在模块执行后发生的断点
Break	B	否	插入一个在模块执行前发生的断点
Bshow	Bs	否	显示特定的模块
Clear	Cl	否	清除一个设置在模块的断点
Continue	C	是	继续进行仿真
Disp	D	是	显示一个模块的输入和输出
Help	H	否	显示调试器命令的帮助信息
Ishow	I	否	使能或禁止积分信息的显示
Minor	M	否	使能或者禁止最小步长模式
Nanbreak	Na	否	设置或清除无穷大值的断点
Next	N	是	跳到下一个时间步的开始
Probe	P	否	显示一个模块的输入和输出
Quit	Q	否	退出仿真
Run	R	否	运行完剩余的仿真
Slist	Slist	否	列出模型的非虚拟模块和模块执行顺序表
States	State	否	显示模型的当前值
Status	Stat	否	显示模型的调试器选项设置
Step	S	是	步进到下一个模块
Stop	Sto	否	停止仿真
System	Sys	否	列出模型的非虚拟系统
Tbreak	Tb	否	设置或清除一个时间断点
Trace	Tr	是	每当模块执行时就显示它的输入和输出
Undisp	Und	是	将一个模块从调试器的显示列表中去掉
Untrace	Unt	是	将一个模块从调试器的 trace 列表中去掉
Xbreak	X	否	在调试器遇到限制步状态时中断
Zcbreak	Zcb	否	在非采样过零点事件发生时中断
Zclist	Zcl	否	列出可能产生非采样过零点的模块

以下是调试命令的详细信息。

(1)ashow

目的　显示代数环。

语法　ashow < gcb | s:b | s#n | clear >

参量　s:b 系统索引是 s，模块索引是 b 的模块；

　　　gcb 当前模块；

s # n 系统 s 标号为 n 的代数环；

clear 清除代数环加亮显示的开关。

说明　ashow gcb 或者 ashow s:b 加亮显示包含指定模块的代数环；ashow s # n 加亮显示系统 s 内的第 n 个代数环；ashow clear 去掉模型图表中代数环的加亮显示。

参见　atrace, slist

(2)atrace

目的　设置代数环的跟踪等级

语法　atrace level

参量　level 跟踪等级 (0 = none, 4 = everything)

说明　atrace 命令设置仿真的代数环跟踪等级，如表 6 - 6 所列。

参见　system, states

表 6 - 6　atrace 命令用法

命　令	显示信息
Atrace 0	没有信息显示
Atrace 1	环路变量的解，求解环路所需的迭代次数，以及估计的解的误差
Atrace 2	同 actrace1
Atrace 3	Atrace2 所显示的信息加上求解环路的雅可比(Jacobian)矩阵
Atrace 4	Atrace3 所显示的信息加上环路变量在每次迭代的瞬时解

(3)bafter

目的　插入一个在模块执行后生效的断点。

语法　bafter gcb

参量　s:b 索引为 s 的系统内的索引为 b 的模块；

gcb 当前模块。

说明　bafter 命令插入由参量指定的模块执行后的断点，当该模块执行后，仿真在此中断。

参见　break, xbreak, tbreak, nanbreak, zcbreak, slist

(4)break

目的　插入模块执行前生效的断点。

语法　break gcb

break s:b

参量　s:b　索引为 s 的系统内的索引为 b 的模块；

gcb　当前模块。

说明　break 命令插入由参量指定的模块执行前的断点，在该模块被执行前，仿真在此中断。

参见　bafter, tbreak, xbreak, nanbreak, zcbreak, slist

(5)bshow

目的　显示指定的模块。

语法　bshow s:b

参量　s:b 索引为 s 的系统内的索引为 b 的模块。

说明　这个命令打开包含指定模块的模型窗口并且选中该窗口。

参见　slist

(6)clear

目的　清除一个模块的断点。

语法　clear gcg

　　　clear s:b

参量　s:b 索引为 s 的系统内的索引为 b 的模块；

　　　gcb 当前模块。

说明　这个命令清除设置在指定模块处的断点，无论是执行前还是执行后。

参见　bafter, slist

(7)continue

目的　继续执行仿真。

语法　continue

说明　continue 命令从当前仿真中断点继续执行仿真，仿真持续执行到它遇上另一个断点和仿真的结束时间。

参见　run, stop, quit

(8)disp

目的　当仿真停止时显示一个模块在停止时刻的输入和输出。

语法　disp gcb

　　　disp s:b

　　　disp

参量　s:b　索引为 s 的系统内的索引为 b 的模块；

　　　gcb　当前模块。

说明　这个命令将指定模块注册为 display 点，无论何时仿真暂停，调试器会在 MATLAB 命令窗口显示所有 display 点在暂停时刻的输入和输出。调用无参量的 disp 命令显示所有的 display 点的列表，而 undisp 命令将指定模块从 display 点列表中去掉。

参见　undisp, slist, probe, trace

(9)help

目的　显示调试器命令的帮助信息。

语法　help

说明　help 命令在命令窗口显示调试器命令的一个列表，该列表包含每个命令的语法和主要描述。

(10)ishow

目的　使能或禁止积分信息的显示。

语法　ishow

说明　这个命令会锁牢仿真过程中积分信息的显示。

参见　atrace

(11)minor

目的　使能或禁止最小步长模式。

语法　minor

说明　minor 命令使调试器进入或退出最小步长模式,在最小步长模式里,step 命令将在一个最小步内单步执行一个模块。在执行完模块排序列表中的最后一个模块后,如果下一个最小步仍然完全处在当前的主时间步范围内,step 命令会把仿真推进到下一个最小时间步,否则,step 命令把仿真推进到下一个主时间步的第一个最小时间步。minor 命令提供了局部细化每个时间步长的能力。

参见　step

(12) nanbreak

目的　设置或清除非有限值断点。

语法　nanbreak

说明　nanbreak 命令使调试器无论何时遇到一个非有限值(上溢或者下溢超出机器的表示范围,或者无穷大的数值),都中断仿真。如果非有限值断点已经设置,那么 nanbreak 命令将清除这个断点。

参见　break, bafter, xbeak, tbreak, zebreak。

(13)next

目的　单步执行到下一个时间步。

语法　next

说明　next 命令执行当前时间步内所有将要执行的模块,然后停止在下一个仿真时间步的开始。执行 next 命令后,调试器加亮显示下一个时间步内将要执行的第一个模块,并且显示下一个时间步的时间。

参见　step

(14)probe

目的　显示一个模块的输入和输出。

语法　probe[<s:b|gcb>]

参量　s:b　索引为 s 的系统内的索引为 b 的模块;

　　　gcb　当前模块。

说明　不带任何参量的 probe 命令将使调试器进入 probe 模式。在 probe 模式下,调试器可以显示用户在模型图表中选中的任何模块的输入和输出。退出 probe 模式,只要输入任何命令即可。Probe gcb 使调试器显示当前选中的模块的输入和输出,probe s:b 则显示由索引 s:b 指定的模块的输入和输出。

参见　disp, trace

(15)quit

目的　退出仿真。

语法　quit

说明　quit 命令终止当前仿真。

参见　stop

(16)run

目的　运行仿真至结束。

语法　run

说明　run 命令从当前的中断点运行仿真至它的结束时间。它忽略断点和 dipslay 点,即不会中断也不会显示 display 点的输入和输出。

参见　continue,stop,quit

(17)slist

目的　列举模型中的非虚拟模块。

语法　slist

说明　slist 命令列出正在调试的模型中的非虚拟模块列表,该列表显示每个模块的名称和模块索引。

参见　systems

(18)states

目的　显示当前的状态值。

语法　states

说明　states 命令显示模型当前状态的一个列表,列表的内容包括状态的取值,状态所属模块的索引和名称。

参见　ishow

(19)system

目的　列出模型中的非虚拟系统。

语法　systems

说明　systems 命令在 MATLABA 命令窗口显示模型中的非虚拟系统,列表会给出系统的名称和它在模型中的索引,这个索引就是用来确定模块索引 s:b 中的 s。

参见　slist

(20)status

目的　显示当前调试选项。

语法　status

说明　status 命令显示了当前的调试选项。

参见　slist

(21)step

目的　单步执行到下一个模块。

语法　step

参量　step 命令执行当前仿真步下一个要执行的模块(此时在模型图表中加亮显示的模块)。执行完 step 命令后,调试器在模型图表中加亮显示下一个要执行的模块以及它的输出信号线,此外,调试器还把这个模块的所属名称作为调试命令提示符的一部分加以显示。

参见　next

(22)stop

目的　终止仿真。

语法　stop

说明　stop 命令将终止仿真，调试器把控制权交还给 MATLAB 命令窗口，即调试命令提示符消失。

参见　run, stop, quit

(23) tbreak

目的　设置或者清除一个时间断点。

语法　tbreak t

　　　tbreak

参量　t 要设置断点的时间值。

说明　tbreak t 命令在指定的时间步设置一个断点，如果一个断点已经存在与指定时间，那么该命令将清除这个断点。如果用户没有指定时间，那么 tbreak 命令在当前时间步设置一个断点。注意这个命令的参量 t 是时间值，而调试器设置断点时是在按时间步来设置的，一般是在大于时间 t，并最接近 t 的仿真时间步设置断点。

参见　break, bafter, xbreak, nanbreak, zcbreak

(24) trace

目的　在模块每次执行时显示模块的输入和输出。

语法　trace s:b

　　　trace gcb

参量　s:b　索引为 s 的系统内的索引为 b 的模块；

　　　gcb　当前模块。

说明　trace 命令将指定的模块注册为 trace 点，调试器在注册模块每次执行时都显示它的输入和输出。

参见　run, stop, quit

(25) undisp

目的　将一个模块从调试器的 display 点列表中去掉。

语法　undisp s:b

　　　undisp gcb

参量　s:b　索引为 s 的系统内的索引为 b 的模块；

　　　gcb　当前模块。

说明　undisp 命令将指定模块从调试器的 display 点列表中去掉。

参见　disp, slist

(26) untrace

目的　将一个模块从调试器的 trace 列表中去掉。

语法　untrace s:b

　　　untrace gcb

参量　s:b　索引为 s 的系统内的索引为 b 的模块；

　　　gcb　当前模块。

说明　untrace 命令将指定模块从调试器的 trace 点列表中去掉。

参见　trace, slist

(27) xbreak

目的　　在调试器遇到限步长状态时中断。

语法　　xbreak

说明　　xbreak 命令使调试器进入 xbreak 模式，即在遇到一个限制解法器采取的仿真时间步长的状态时，调试器会中断模型的执行。如果 xbreak 模式已经开启，那么 xbreak 命令将关掉这个模式。

参见　　break，bafter，zcbreak，tbreak，nanbreak

(28)zcbreak

目的　　设置非采样过零点事件断点。

语法　　zcbreak

说明　　zcbreak 命令使调试器进入过零点中断模式，即调试器在非采样过零点事件发生时中断仿真，如果过零点中断模式已经开启，那么 zcbreak 命令将关闭该模式。

参见　　break，bafter，xbreak，tbreak，nanbreak，zclist

(29)zclist

目的　　列出可能包含非采样过零点的模块。

语法　　zclist

说明　　zclist 命令在 MATLAB 命令窗口，显示有可能发生非采样过零点事件的模块的列表。

参见　　zcbreak

第7章 仿真运行和结果分析

建好一个模型之后,下面的事情就是运行模型和分析仿真的结果。前面的章节对运行仿真,只是粗略地提了一下,这一章对它进行详细的讲解。

7.1 使用菜单命令运行仿真

使用菜单命令运行仿真十分简单,而且用户和模型之间交互能力强。这些命令允许用户选择常微分方程(ODE)解法器和改变仿真的参数,而无需去记忆 MATLAB 的命令语法。更为重要的是,这种方法使得用户在仿真运行中可以进行某些操作,它们是:

(1)可以修改许多仿真参数,包括仿真结束时间、解法器和最大步长;

(2)可以同时仿真其他系统;

(3)可以单击模型的连线,通过浮动的 scope 或者 display 模块来查看该直线传递的信号取值;

(4)可以修改模块的参数,只要这种修改不会导致下面一些量的变化:模块状态、输入或者输出的数目,采样时间,过零点的数目,模块参数的向量长度,内部模块工作向量的长度。

在仿真运行中,Simulink 不允许对模型结构进行任何修改,诸如添加或者删除一个模块、一条连线等等。要进行这些修改,必须先终止仿真,修改完后,再运行仿真。

用菜单命令运行仿真的方法,其实在前面的学习中读者已经接触了,例如,从 simulation 菜单下选择 start 命令,或者使用工具栏上的运行快捷按钮。但前面所用到的那些操作都是很不完整的,运行一个仿真完整的过程应该要分成三个步骤:设置仿真参数,开始仿真和查看结果。

1. 设置仿真参数和选择解法器

设置仿真参数和选择解法器,选择 simulation 菜单下的 Parameters 命令。选择这个命令之后,Simulink 会显示一个仿真参数对话框,它用三个页面来管理仿真的参数:

(1)Solver 页,它允许用户设置仿真的开始和结束时间,选择解法器,说明解法器参数以及选择一些输出选项;

(2)Workspace I/O 页,作用是管理模型从 MATLAB 工作空间的输入和对它的输出;

(3)Diagnostics 页,允许用户选择 Simulink 在仿真中显示的警告信息的等级。

本章的下一节将会对这些仿真参数设置参数页面进行详细的说明。

仿真参数可以被设置为有效的 MATLAB 的表达式,包括常数、工作空间变量名、MATLAB 函数和数学运算符。

设置完参数后，就可以按下对话框下的 apply 按钮以应用你的设置，若要同时关闭按钮，则选择 OK 按钮。如果是按下 Cancel 按钮，则会取消前面所作的任何修改。

2. 开始仿真

做好上面的工作之后才是开始仿真，除了我们已经用到的两种方法，还可以用快捷键 Ctrl+T 来完成这个工作。提前结束仿真，同样可以在 simulation 菜单和工具栏找到相应的命令，又可以使用与开始仿真相同的快捷键。此外，Simulink 还允许用户暂停仿真，这些命令都在 simulation 菜单里面，它们在仿真运行后会自动出现，读者可以建立一个简单的模型来试验这些命令，在试验时，建议读者先把仿真时间设置长一些。

3. 仿真诊断对话框

如果仿真过程中有错误产生，Simulink 会中断仿真并在仿真诊断对话框里显示错误信息。

图 7-1 就是一个诊断对话框示例。它分为两个面板，上面的面板显示了每一个错误的信息，这些信息有：

- Message——信息类型(例如，模块错误、警告、log)；
- Source——导致错误的模型元素(连线或模块等)的名称；
- FullPath——导致错误的元素的路径；
- ReportedBy——报告错误的组件(如 Simulink、rtw)；
- Summary——错误信息的简写，便于在列表里显示。

而 Simulink 会在对话框下面的面板，显示错误信息的完整内容，注意如果具有多个错误信息，显示在下部面板的信息和上部面板被选中的错误信息相对应。Simulink 除了能显示这些信息，还可以打开模型的图表，并把错误单元（模块或者直线）用黄色来表示。用户还可以双击上部面板的错误信息，Simulink 就会显示出现错误的具体位置，如上例中，就会显示出 Gain 模块的参数设置对话框。另外一种替代的方法是 open 命令。

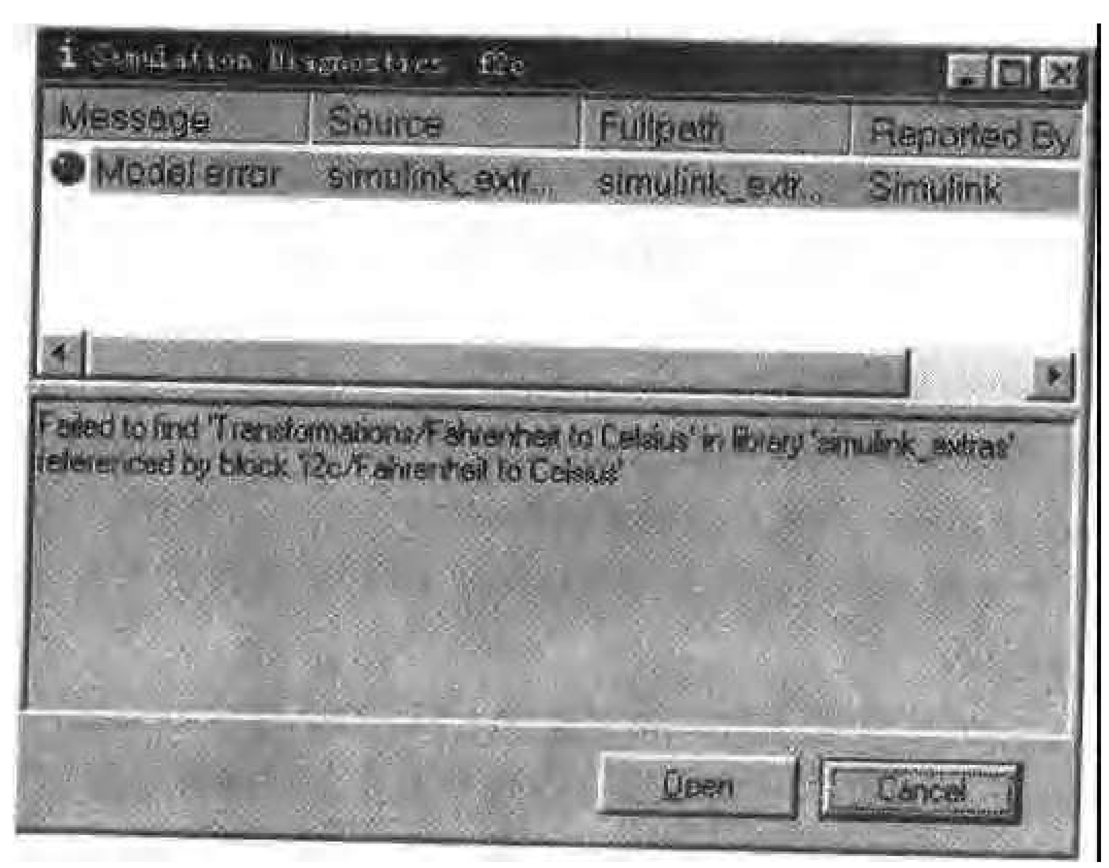


图 7-1 仿真诊断对话框

7.2 仿真参数对话框

在 Simulink4.0 里，仿真参数对话框分为 Solver 页、Workspace I/O 页、Diagnostics 页和 Advanced 页。它和 Simulink3.0 的区别在于增加了一个 Advanced 页，其余的变化则不是太大。

7.2.1 Solver 页

在 solver 页，如图 7-2 所示，可以进行的设置有：选择仿真开始和结束时间；选择解法器，并设定它的参数；选择输出选项。

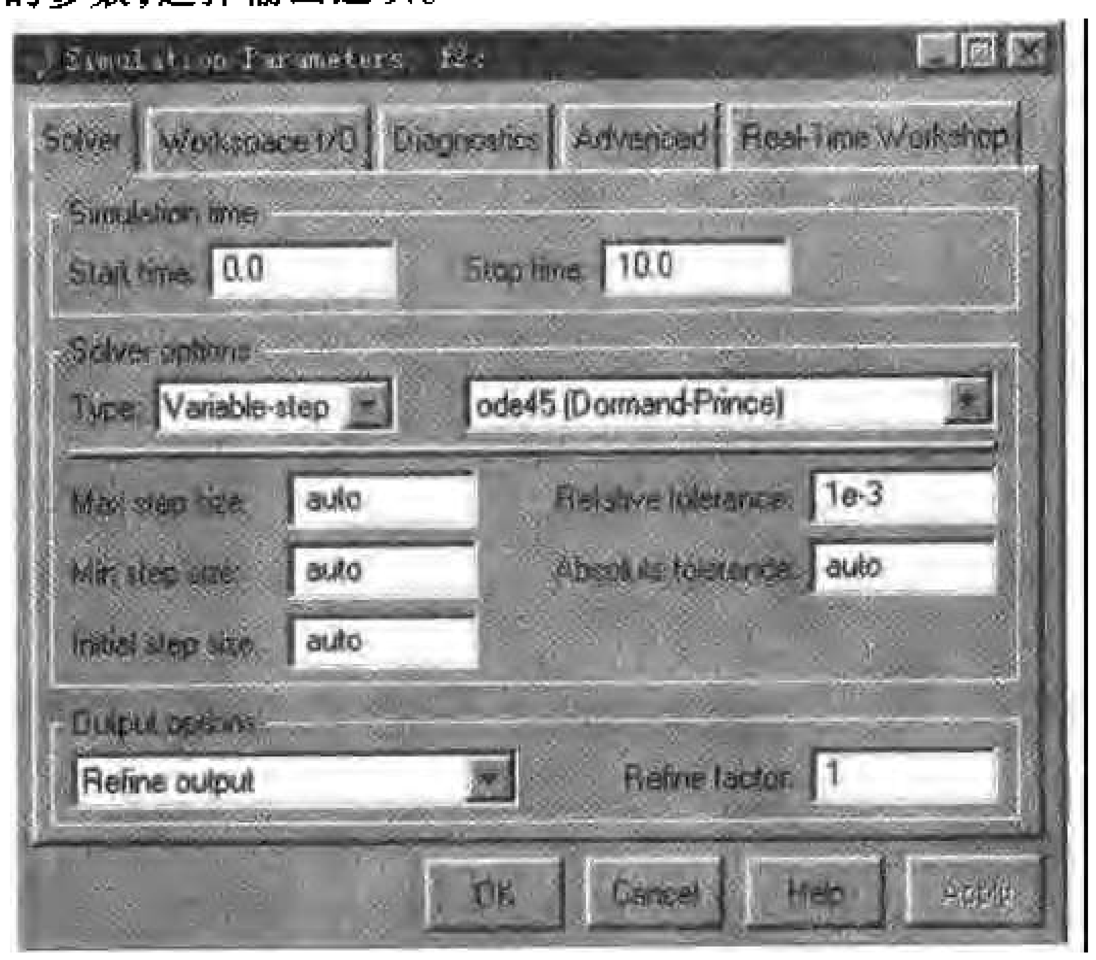


图 7-2　Solver 页

1. 仿真时间

用户可以设置仿真的开始时间和结束时间。注意，这里的时间和真实的时间并不一样，这里的时间只是计算机模拟中对时间的一种表示。比如 10 s 时间，如果采样步长定为 0.1，则需要执行 100 步，若把步长减小，则采样点数增加，那么实际的执行时间会增加。总的说来，执行一次仿真要耗费的时间依赖于很多因素，包括模型复杂度、解法器步长和计算机时钟速度。

2. Solver

Simulink 模型的仿真综合了常用的几种常微分方程，Simulink 提供了一些用于这些方程仿真的解法器。由于动态方程的差异性，所以有些解法器解决某些特定的问题会比另外一些解法器有效。因此，要获得精确而快速的仿真结果，必须仔细地选择解法器和设置它们的参数。

首先是根据仿真步长来进行选择，这时有两个选项：定长解法器和变长解法器。变长解法器可以在仿真过程中改变步长。这类解法器提供误差控制和过零检测。定长解法器

在仿真过程中提供固定的步长，它们不进行误差控制和定位过零点。

(1)缺省解法器(Solver)

如果用户没有选择解法器，那么 Simulink 会根据你的模型是否有连续状态，自动选择一个。这可以分为两种情形：

(a)如果模型有连续状态，就选择 ode45，它是一种性能良好的通用解法器。然而，如果用户知道所仿真的模型是 stiff 问题，并且尝试 ode45 后不能获得可接受的结果，用户可以试试 ode15。

(b)如果模型没有连续状态，那么 Simulink 使用称为 discrete 的变长解法器，并且显示一个信息表明模型现在使用的解法器不再是 ode45。注意 Simulink 除了提供一个称为 discrete 的变长解法器，也提供了一个称为 discrete 的定长解法器。

现在请读者更改解法器的设置。首先是使用 variable - step 解法器，请在右边的列表框选择 discret；设置完后，执行模型。注意到这个模型的 out 模块会把结果输出到 MATLAB 工作空间。读者可以用 who 查询 MATLAB 工作空间中存在的变量，会发现增加了两个变量 tout、yout。

> > tout

对应定长解法器，输出的仿真时间为：

tout = [0　0.2500　0.5000　0.7500　1.0000　1.2500　1.5000　1.7500　2.0000　2.2500　2.5000　2.7500　3.0000]

对于变长离散解法器，输出的仿真时间为：

[0.0　0.5　0.75　1.0　1.5　2.0　2.25　…]

从上面的结果不难看出，定长解法器的步长是基准采样时间，而变长离散解法器采取允许的最大步长。

(2)变长解法器

在 Simulink 参数对话框的 solver 页列出的变长解法器有：ode45，ode23，ode13，ode15s，ode23s 和 discrete；其中 ode45 是有状态的连续系统和无状态的离散系统的缺省设置。

(a)ode45 基于显式的 Runge - Kutta(4,5)(龙格 - 库塔)公式——Dormand - Price-pair。它是单步解法器，也就是在计算 y(tn)时，它仅需要最近处理时刻的结果 y(tn - 1)。一般来说，面对一个仿真问题，最好是首先试试 ode45。

(b)ode23 同样是基于 Bogacki 和 Shampine 的 Rung - Kutta(2,3)的显式对。它在误差限要求不高和求解的问题不太难的情况下，可能会比 ode45 更有效。Ode23 也是一个单步解法器。

(c)ode13 是一种阶数可变的 Adams - Bashforth - Moulton PECE 解法器。它在误差容许要求严格的情况下通常比 ode45 有效。Ode13 是一种多步解法器，也就是在计算当前时刻输出时，它需要以前多个时刻的解。

(d)ode15s 是一种基于数字微分公式的解法器(NDFs)。它们和后向微分公式有联系但要比后者有效。和 ode13 一样，ode15s 也是一种多步解法器，如果用户估计要解决的问题是比较困难的，或者不能使用 ode45，或者即使使用效果也不好，就可以用 ode15s。

(e)ode23s 基于修改二阶 Rosenbrock 公式。因为它是一个单步解法器，所以在弱误

差允许下的效果好于 ode15s。它能解决某些 ode15s 所不能有效解决的 stiff 问题。

(f)ode23t 是梯形规则的一种自由插值实现。这种解法器适用于求解适度 stiff 的问题而用户又需要一个无数字震荡的解法器的情况。

(g)ode23tb 是 TR－BDF2 的一种实现，TR－BDF2 是具有两个阶段的隐式 Runge－Kutta 公式，它的第一个阶段是一个 trapezodial 法则，第二个阶段是二阶的后向差分公式。在实际上，使用相同的循环矩阵来估算两个进程。

(h)discrete(变长)，当 Simulink 检查到模型没有连续状态时使用它。

(3)定长解法器

可以选择的定长解法器包括 ode5、ode4、ode3、ode1 和 discrete，用法说明如下：

(a)ode5 是 ode45 的固定步长版本，Dormand－Prince 公式；

(b)ode4 是 RK4，四阶 Runge－Kutta (龙格－库塔)公式；

(c)ode3 是 ode3 的固定步长版本，Bogacki－Shampine 公式；

(d)ode2 是 Heun 方法，也称为改进欧拉公式；

(e)ode1 是欧拉方法；

(f)discrete (fixed－step)是一个实现积分的定长解法器。它适合没有状态的模型，而且过零点检测和错误控制对模型不重要。

如果觉得仿真的结果不满足工程要求，可以参阅本章的"提高仿真性能和精确性"一节。

3. 解法器选项

缺省的解法器选项对大多数问题都可以获得精确和有效的结果，但是在有些情况下，调整参数可能获得更好的结果。

4. Step size

对于变长解法器，用户可以设置最大的和推荐的初始步长参数，缺省情况下，步长自动地确定，它由值 Auto 表示。

对于定长解法器，用户可以设置固定步长，它的缺省值同样是 Auto。

(1)Maximum step size(最大步长参数)

最大步长参数决定了解法器能够使用的最大时间步长。它地缺省值由下面的公式确定：

$$h_{\max} = \frac{t_{\text{stop}} - t_{\text{start}}}{50}$$

一般来说，缺省值就足够了，如果用户担心漏掉关键的行为，那可以设置这个参数以防止解法器采用过大的步长。提供仿真的时间跨度很长，那么缺省步长对解法器找到解而言就太长了。提供用户的模型具有周期性的行为并且用户知道这个周期，那么就可以把最大的仿真步长设置为周期的 -个系数。

(2)Initial step size

初始步长参数，缺省情况下，解法器选择一个初始步长在仿真开始时刻检查状态的微分。如果第一个步长太长，解法器将会漏掉重要的行为。Initial step size 是建议的初始步长，解法器使用该步长，如果不能满足误差标准，就减小步长。

(3)Minimum step size

设定解法器能过采取的最小时间步。如果解法器需要采用一个更小的仿真步来满足误差限度的要求，它会给出一个警告表示当前有效的相对误差限度。这个参数可以是一个大于零的实数值，也可以是一个两个元素的向量。向量的第一个元素表示最小步长的大小，而第二个元素表示 Simulink 在给出错误提示之前，最小步长误差提示所能出现的最大次数。比如，向量的第二个元素设为 0，表示在解法器第一次必须采取小于设定最小值的仿真步时，就产生错误。这等效于将 Diagnostics 页的最小步长冲突诊断错误选项置为 on 的状态。将第二个元素设置为 -1，表示警告能够产生的次数没有限制。这个值也是当最小步长设置的值是标量时的缺省值。最小时间步参数元素的缺省值是机器的表示精度和 -1(警告次数不限)。

5.Erroe Tolerances(误差限度)

解法器采用标准的当前误差技术来监控每一个时间步的误差。在每一个时间步期间，解法器在该步结束时，计算状态值，并决定当前误差——这些状态值的估计误差。解法器然后把局部误差和可接受的误差相比较，后者是相对限度(rtol)和绝对限度(atol)的函数。如果有一个状态的当前误差大于可接受误差，解法器就会减小步长，并重新开始计算。

这里相对限度和绝对限度的区别是，相对限度是指误差相对于状态的值，是一个百分比，缺省值是 1e-3，表示状态的计算值要精确到 0.1%。而绝对限度表示误差值的门限，或者说在状态值为零的情况下，可以接受的误差。

知道了相对限度和绝对限度，就可以按下面的公式计算出可接受恶误差。

$$e_i \le \max\{ rtol * |x_i|, atol_i \}$$

如果 absolute tolerance 被设成了 auto，那么 Simulink 就为每一个状态设置初始 absolute tolerance 为 1e-6。当仿真进行时，Simulink 重新设置每个状态的 absolute tolerance 为最大的值。例如，如果一个状态从 0 变到 1，并且它的 rtol 为 1e-3，则在仿真的末尾，该状态的 atol 就会变成 1e-3。如果是从 0 变到 1000，相应的最终 atol 是 1。

这样计算得到的 atol 设置有可能不合适，那用户可以自己确定一个合适的值。最简单的方法，用户可以枚举不同的 absolute tolerance 取值，分别用它们进行仿真，然后在其中确定一个结果最好的参数。

6.Mode(模式)

在用户选择了 fixed-step 解法器时，solver 页会显示一个 mode 列表，它里而的选项有：MultiTasking、SingleTasking 和 Auto。它们的意义分别如下：

(1)MultiTasking。选择这种模式时，当 Simulink 检测到模式间有非法的采样速率转换，它会给出错误提示。所谓的非法采样速率转换是指两个工作在不同采样速率的模块之间的直接连接。在实时多任务系统中，如果任务之间存在非法采样速率转换，那么就有可能出现一个模块的输出在另一个模块需要时却无法利用的情况。通过检查这种转换，MultiTasking 将有助于用户建立一个符合现实的多任务系统的有效模式。

使用速率转换模块可以减少模型中的非法速率转换。Simulink 提供了两个这样的模块：Unite Delay 模块和 Zero-Order 模块。对于从慢速率到快速率的非法转换，可以在慢输出端口和快输入端口插入一个 Unite Delay 模块。而对于快速率到慢速率的转换，则可以插入一个 Zero-Order Hold 模块。

(2)SingleTasking 模式。这个模式不检查模块间的速率转换,它在建立单任务系统模型时常用,在这种系统就不存在任务同步问题。

(3)Auto 模式。这种模式下,Simulink 会根据模型中模块的采样速率是否一致,自动决定切换到 multitasking 和 singletasking。如果模型中所有模块工作在相同的采样速率,那么就使用 singletasking 模式,反之工作在不同的采样速率,则使用 multitasking 模式。

7. 输出选项

solver 页的输出选项域允许用户控制 Simulink 产生的输出。它包括三个选项:Refine Output(精细输出),Produce additional output(产生额外输出)和 Produce specified outpu tonly(只产生指定的输出)。

(1)Refine output 选项。这个选项可以理解成精细输出,其意义是在仿真输出太稀松时,Simulink 会产生额外的精细输出,这一点就像插值处理一样。用户可以在 refine factor 设置仿真时间步间插入的输出点数。例如设置为 2,则在仿真采样时间点间产生 2 个额外输出。

产生更光滑的输出曲线,改变精细因子比减小仿真步长更有效。当精细因子变化时,Simulink 会根据一个扩展公式来计算在这些点的输出,但不会改变解法器仿真时采用的步长。精细输出只能在采用变长解法器时才能使用,并且在 ode45 效果最好。ode45 可以采用大的仿真步长。当用图形显示仿真输出时,就会发现曲线有时会很不光滑,可以通过增大精细系数来解决这个问题,一个典型的值是 4。

(2)Produce additional output 选项。它允许用户直接指定产生输出的时间点。一旦选择了这个选项,那么在它的右边会出现一个 output times 编辑框,在这里用户指定额外的仿真输出点,它既可以是一个时间向量,也可以是表达式。与精细因子相比,这个选项会改变仿真的步长。

(3)最后一个选项是 Produce specified output only。它的意思是让 Simulink 只在指定的时间点上产生输出。为此解法器要调整仿真步长以使之和指定的时间点重合。这个选项在比较不同的仿真时可以确定它们在相同的时间的输出。

下面举一个例子来说明这三个选项的不同。

假定一个仿真在以下时间点产生输出:

{0 2.5 5 8.5 10}

当选择 refine output,并且它的精细因子为 2 时,产生输出的时间点为

{0, 1.25, 3.75, 5, 6.75, 8.5, 9.25, 10}

如果选择 Produce additional output 项,并且说明指定时间点为

```
>> [0:10]
```

则除了在 0、1、2、3、4、5、6、7、8、9、10 外还要在 2.5、8.5 产生输出,但如果选择 only,则只有 0 到 10。

7.2.2 Workspace I/O 页

这个页面的作用是定义将仿真结果输出到工作空间,以及从工作空间得到输入和初始状态。图 7-3 是它的全貌。

Workspave I/O 页被分成了三个面板:Load from workspace、Save to workspace 和 Save

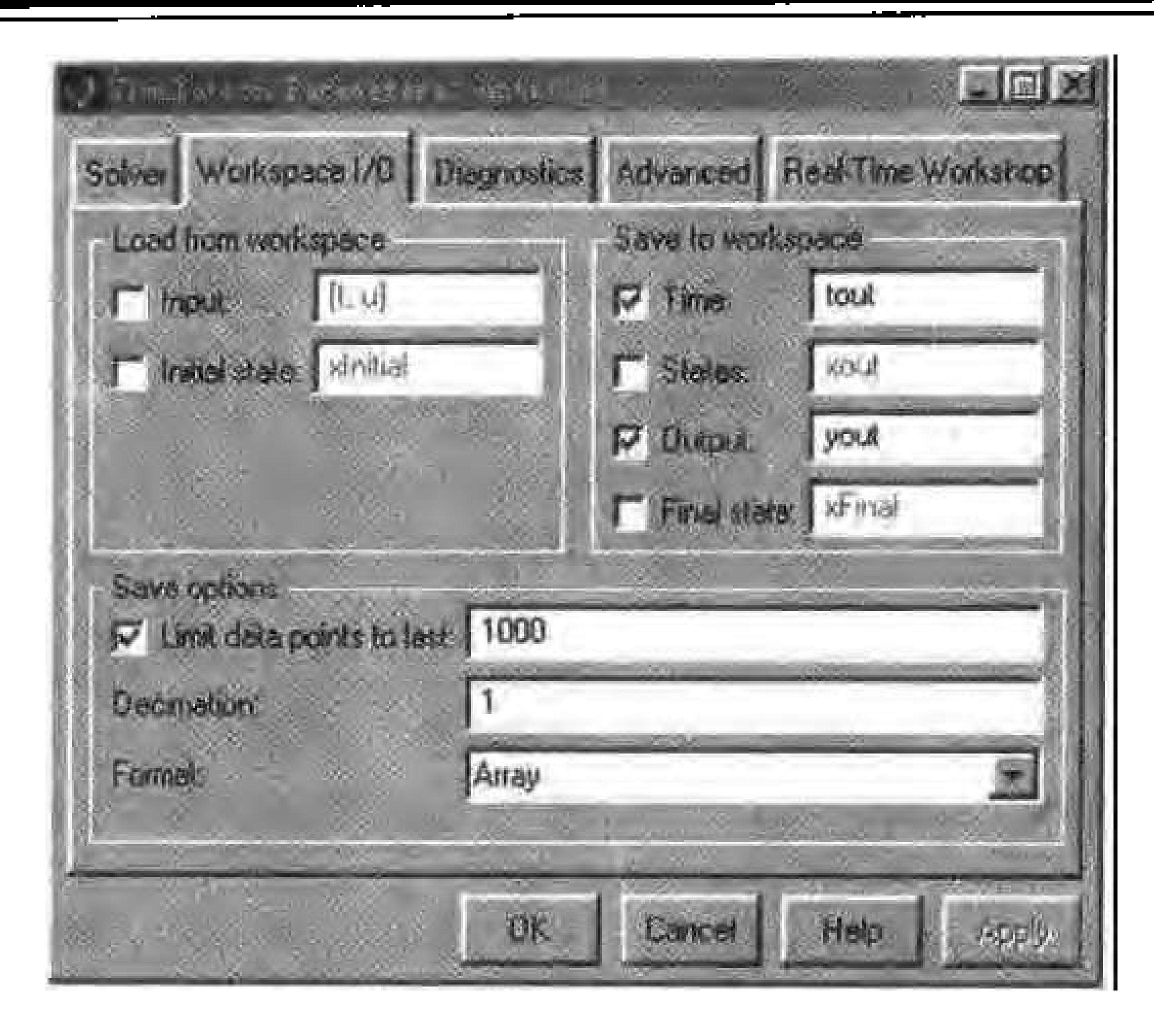


图 7-3 Workpace I/O 页

options。它们各自的分工是:Load from workspace 负责从工作空间获得输入和初始状态,Save to workspace 负责保存变量到 MATLAB 工作空间,而 Save options 则是让用户保存到工作空间的变量的格式。Workspace I/O 页主要以检查框为主要的控制风格,用户通过是否选中检查框来设置 Simulink 是否执行该选项所对应的功能。例如选择了 Load from workspace 的 input 项,就表示要从 MATLAB 工作空间获得输入,而后面的编辑框则是包含输入值的变量名,只有在 input 被选中时,用户才能编辑变量名称。Save to workspace 面板也是一样,用户只有通过检查框来确定是否输出这一项,然后才能自定义包含它的变量名。

1. 从 MATLAB 工作空间引入输入

单从参数的设置而言,从 MATLAB 工作空间获得输入并没有什么特别的地方,但是在运用中一定要注意输入变量的格式。

可以使用的外部输入格式有以下几种:外部输入矩阵、带时间的结构、结构和与端口对应的结构。

下面我们来举一个从工作空间输入的例子来一一说明这些结构,如图 7-4 所示。

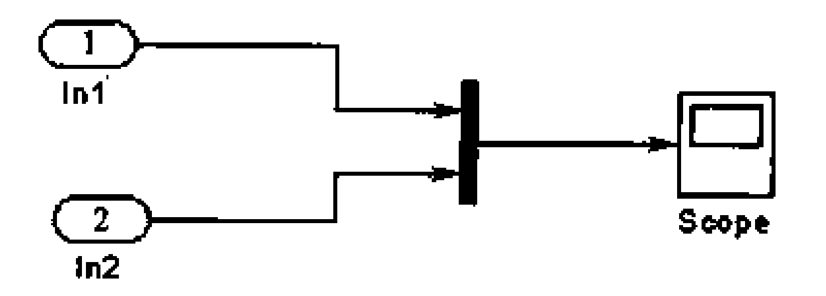


图 7-4 从工作空间输入示例

图 7-4 是一个简单的模型,它没有信号源,只有两个输入端口,in1 的输入宽度是 1,而 in2 的宽度是 2。这个模型的功能是从 MATLAB 工作空间获得输入,并显示在 scope

里。

首先来看看外部输入的矩阵格式。请按照图 7 – 5 设置仿真参数的 workspace I/O 页，其实只需选中 input 就可以了，因为这时的缺省格式就是[t，u]。

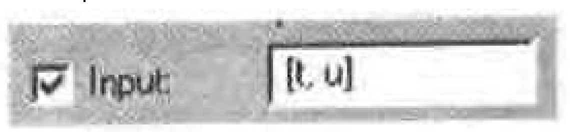


图 7 – 5　参数设置

图中的形式定义了一个矩阵，时间是系统的第一列，而输入 u 表示剩下的列，这就是所谓的外部输入矩阵格式。输入矩阵的第一列必须是升序的时间向量，这和仿真的时间演变顺序是一致的。而矩阵的其余列则是输入向量，每一列对应一个端口的输入，排列的顺序根据端口序数来确定，例如第一列就对应第一个输入端口。如果某个端口的信号宽度是大于 1，那它就对应着相应数目的列。于是矩阵的每一个行就包含了某一次仿真时间点，以及当时的输入值。接着，就要在 MATLAB 工作空间定义变量 t 和 u 了。例如

```
>> t = (0 : 0.1 : 10 )';
%定义时间变量 t，注意'的作用是转置，把行向量变为列向量
>> u = [ sin (t), 4 * cos (t) ];
%定义输入变量，因为输入端的总宽度是 3，所以要定义 3 列。
```

定义完这些变量后，就可以运行仿真了，读者会发现在 scope 模块显示了这三条曲线。如图 7 – 6 所示。

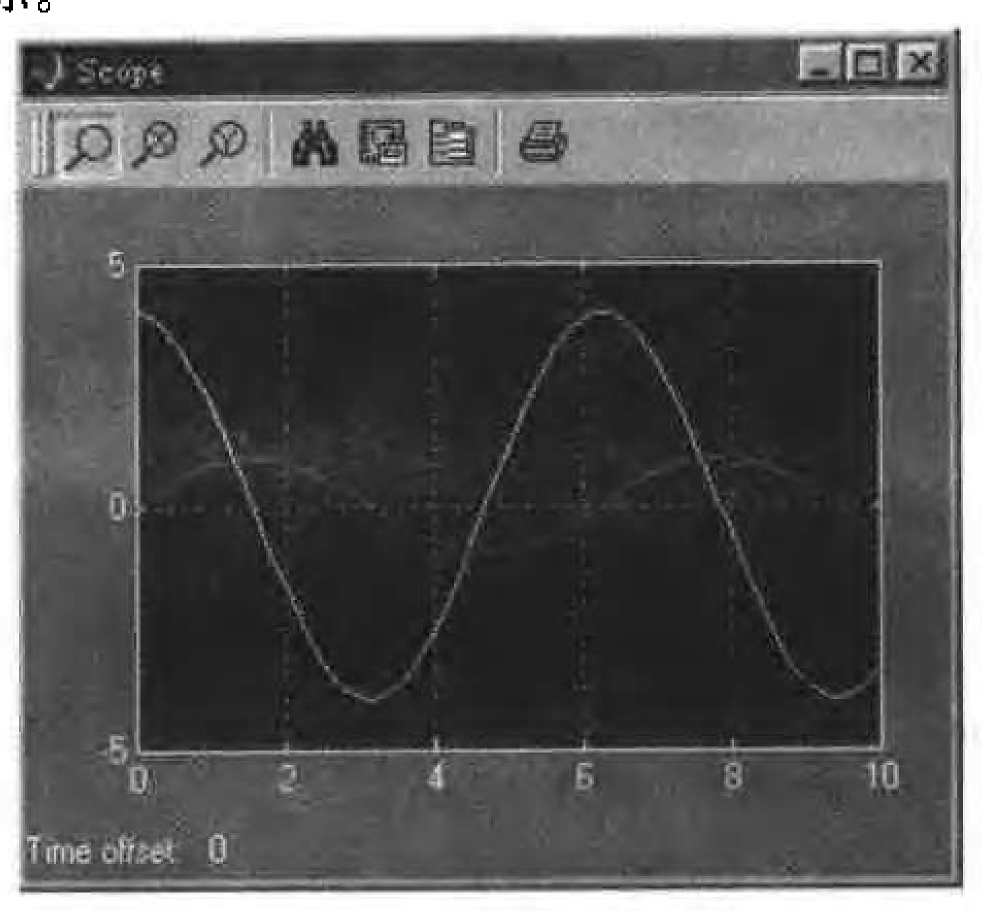


图 7 – 6　scope 模块显示的曲线

在定义外部变量时，要注意以下几点：

(1)外部输入矩阵的列数是所有需要从工作空间获得输入的端口的信号数(就是它们的宽度和)再加上 1。

(2)存在多个端口时，输入信号与端口的对应是依赖端口序数的。例如上面模型中，端口 1，in1 获得的输入是 cos(t)，而 in2 获得的输入是[sin (t)，4 * cos (t)]，它的宽度是 2。

Simulink 还可以从工作空间的 struct 格式的变量读取数据，为此要先在 input 的临近的编辑框设置结构名，如图 7－7 所示，“a”是本例中为外部输入结构的名字，读者当然可以取别的名称。

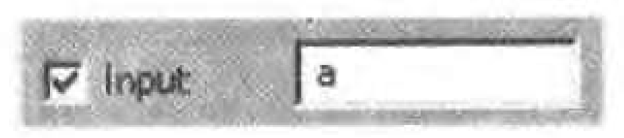


图 7－7 结构变量名的设置

下面的问题就是如何在 MATLAB 工作空间定义结构 a。

```
> >a.time = (0:0.1:10)' ; %定义 struct a 的 time 字段
> >a.signals (1) .values = [ cos (t) ] ;
%t = (0:0.1:10), 在前面示例中已经定义,故省略这一步
> >a.signals (2).values = [ sin (t) , 4 * cos (t) ];
```

这里，读者请记住 MATLAB 里定义变量不需要事先进行类型定义，a.time 就表示了新建立的变量是一个结构，time 是其中的一个字段，而不需要先定义 a 是一个结构类型的变量，它包括一个字段 time(在许多高级语言，如 C 等都是需要预先进行定义声明)。关于结构的详细用法请参阅 MATLAB 的帮助文档。

作为模型外部输入变量的结构必须具有特殊的格式，Simulink 是这样规定的：

它必须具有两个字段，time 和 signals。Time 字段保存代表仿真时间的向量，这就是“具有时间的结构”这一名称的来由。而 signals 的结构比较复杂，它是一个以结构作为元素的数组，数组的每一个元素对应一个输入端口。面每个元素都是一个包含 values 字段的子结构，values 字段包含相应端口的输入值。如示例模型中，有两个字段，所以 signals 数组的大小是 1×2，而对于端口 2，因为它的宽度是 2，所以它对应的输入值是两列的矩阵。读者可以在 MATLAB 子空间出现 a，看看它的组成。

```
> > a %查询结构 a
a =

    time :[101 x 1 double]
    signals:[1 x 2 struct]
> >a.signals              %查询 signals 字段
1 x 2 struct array with fields :
      values
> >a.signals (1).values;     %这个命令就查询端口 1 的输入值
```

注意，在定义作为外部输入的结果变量时，一定要注意各个字段的拼法，它们是 time，signals 和 values，有时候可能会写错大小写，或者是少了 s，这都会导致运行出错。因为 MATLAB 是解释性的语言，它只会按照用户的意愿添加相应的字段，而不会提示出错。然而在 Simulink 从结构变量取相应的字段时，找不到该字段，就会出现错误。而且，Simulink 也会关心该变量是否有它不需要的字段，于是可以不用重新清除结构变量、就可以直接添加正确的字段进去。当然最好的方法是用 clear 清除原来的变量，再重新定义。

以上三种可以被 Simulink 识别的外部变量形式是结构 struct，它与具有时间的结构

的区别就在于前者的 time 字段是空的,即

```
> >a.time = [  ] %假定外部输入变量还是 a
```

至于 signals 字段的定义和后者一模一样。

也就是说模型不需要从外部获得仿真时间的输入,而是由在模型内的采样时间定义。但是输入却是从 MATLAB 工作空间获得,获取的顺序是,在第一个仿真时间步就取该端口对应输入向量的第一个元素,第二个仿真时间步取第二个元素,依此类推。

还是以上面的模型作为例子,显然,在仿真参数对话框的设置和带时间的结构形式一样,可以不用去改变。只要改变 a 在 MATLAB 子空间的定义就可以了。为此,请先清除变量 a 以前的定义(当然也可以在仿真参数对话框改用别的变量名称,从而不需要清除)。

```
> >clear a;
> >a.time = [];
> >a.signals (1).values = [cos(1)];
> >a.signals (2).values = [sin(t), 4 * cos (t) ];
```

而为了运行模型还必须对模型两个输入端口的参数进行改动。请双击输入端口,把它们的 sample time 都改成 0.1,并去掉对话框中对 intoplate 检查框的选中。这样设置后运行模型,就不会由错误了。这里一定要把输入数据和仿真时间步的对应关系弄清楚。例如,分别把端口 1 和端口 2 的采样时间设为 0.2 和 0.05,得到的运行结果为显示波形相应的频率变为了原来的 2 倍。

这种现象,很容易理解。因为在 MATLAB 工作空间的输入变量设定的输入值只有 100 个采样值,而在输入到模型时,Simulink 按照自己的采样时间步长依次把采样值输入。于是,对于输入端口 1,它的采样时间是 0.2,于是 100 个采样值就可以持续 20 s,所以显示在 scope 的图形实际是 cos(t/2),而不是 cos(t)。而对于输入端口 2,它的采样时间是 0.05,输入的 100 个采样点只能持续 5 s 的时间,对于剩下的仿真时间,Simulink 只能把它当成 0 来处理,显示波形相应的频率变为了原来的 2 倍。

第四种被 Simulink 认可的外部数据输入格式是与端口对应的结构。这种格式,为模型的每一个端口设定一个外部输入变量,而每一个变量都是一个结构的变量,它们可以是带时间的,也可以是不带时间的。和前面的两种一样,这些结构都有一个 signals 字段,它里面就包含该端口的输入数据。在设置仿真参数时,需要把所有的结构变量按照所对应的端口的序数排列写在编辑框里。图 7-8 说明了如何对前面的示例模型定义这种格式。



图 7-8　参数设置

这就是定义对于输入端口 1 的外部变量是 d1,端口 2 对应的外部输入变量是 d2。下面分别把 d1 和 d2 设置为带时间和不带时间的结构。

```
> >t = (0:0.1:10)';
> >d1.time = t;
> >d1.signals.values = (t);
> >d2.time = [];
```

```
>> d2.signals.values = [sin (t), cos(4 * t) ];
```

为此,请读者设置好各个端口的采样时间,一般而言,端口1的外部输入变量带时间,所以可以把它的采样时间参数设为-1,当然读者给它指定也可以,这时Simulink就以用户指定的为准。而对于外部输入变量不带时间的端口2,则必须设定一个采样时间,而且不能把它的值设为0.因为那是连续系统的采样时间。

2. 保存输出到工作空间

至于保存输出到MATLAB工作空间,它要涉及到两个面板。Simulink允许用户在Save to workspace选择要返回到MATLAB工作空间的信号以及保存该量的变量名。它的设置形式和设置与外部输入基本上一样,首先是选中检查框,然后再指定返回变量名称。在面板上可以选择的有:time、states、output和state。其中,time和output是缺省被选中的,一次一般运行一个仿真模型后,在MATLAB工作空间都会增加两个变量tout和yout,而至于states和final state的概念读者在学习了S-Function的编写之后就会很清楚了。而save option的一些参数,是用来说明返回变量的行数,选中时由后面的编辑框来设置具体的数目。Decimation参数指定了一个亚采样因子,它的缺省值是1,也就是对每一个仿真时间点产生值都保存,而对于2,则是每隔一个保存。至于format参数,它则是用来说明返回数据的格式。可以选择的格式有:matrix、structure和structure with time。它们的定义和前面的外部输入数据相应格式的定义是相同的。这里就不再细说了。

3. Loading and Saving States

可以保存和载入的状态,有几种类型:初始状态、缺字状态和最终状态。所谓初始状态是指在仿真开始时各个模块的状态,相应的最终状态则指仿真结束时的状态。通过设置这里的参数可以从MATLAB工作空间导入初始状态。而通过把最终状态保存在变量里,就可以使得这些状态能应用于其他的仿真。这个特性在用户想保存一个稳定状态并且在已知的状态重新开始仿真时,非常有用。同输入或者输出一样,Simulink能够识别的状态变量格式(导入时),对能生成的格式(保存)也有特殊的要求。在导入初始状态时,相应的变量就要符合Simulink的要求,而在保存状态时,用户可以在save option面板的format参数来制定返回的格式。在名称上,状态变量的格式和前而所讲的是很类似的,也是matrix、struct with time和struct。除了matrix格式差不多外,应用于状态时,另外两种格式还是有它们特殊的要求。

struct with time格式,是带时间的结构格式,它有两个最顶层的字段:time和signals。其中time的定义和前面的一样,都是仿真时间向量。这里的signals字段也是一个子结构的数组,数组里的每一个子结构都对应这模型中存在状态的一个模块。但这里的子结构要比前面所讲的复杂得多,这种复杂性的增加是必须的,也是可以理解的。因为Simulink在读取这些子结构时,要判断它对应于哪个模块,这和输入端口不同,你无法事先对模块编号,因此要开辟一个专门的字段来标识模块名。事实上,每个子结构都有三个字段:values、label和blockName。value字段包含该模块所有的状态向量值,而label可以取值为Cstate或者Dstate_n,n取值从1一直到相应名具有的离散状态集的最大数目,而blockname字段标识了该子结构所对应的模块名,这个名称在模型图表中就是模块的标签。

而struct格式,则是struct with time格式time字段为空时的特殊情况。

在initial states检查框没有被选中时,Simulink使用模块参数里initial conditions参数

设置取值。

当模型是多状态模型时，用户要设置初始状态，就有一个确定状态的次序的问题。下面的命令可以做到这一点。

```
>>[ sizes, x0, xstord ] = sys ([], [], [], 0 ) ;
```

其中，sys 代表模型的名称，也就是保存模型的模型文件的名称。三个返回变量的意义又如下：

sizes，反映了模型的某些特性的一个向量，只有前两个元素和初始状态有关，sizes(1)代表连续状态的个数，sizes(2)代表离散状态的个数。

x0，模块的初始条件。

xstord，一个包含模型中所有具有状态的模块的完全路径名称的字符串矩阵，xstord 中模块的顺序和 x0 相同。

例如，要获得 Simulink 提供的演示模型 vdp 的初始状态，可以使用如下的命令。

```
>>[ sizes, x0, xstord ] = vdp ([], [] [], 0)

size =
      2
      0
      2
      0
      0
      0
      1

x0 =
      2
      0

xstord =
      'vdp/x1'
      'vdp/x2'
```

返回的这些变量所透视的信息为，vdp 模型有两个连续的初始状态，没有离散初始状态，模型中有状态的模块为 x 1 和 x 2 模块，它们的初始条件分别为 2 和 0。

7.2.3 Diagnositics 页

图 7－9 是 Diagnostics 页的全貌，读者可以按自己的兴趣去摸索上面的一些设置。Diagnostics 页分成两个部分，仿真选项和配置选项。配置选项下的列表框主要列举了一些常见的事件类型，以及当 Simulink 检查到了这些事件的处理：什么也不作——none，还是发出警告信息——warning，或者给出错误提示——error。对于这些事件的处理方式，用户可以按自己的意愿设置。方法很简单，先用鼠标左键单击要设置的事件类型，然后在右边的单选按钮

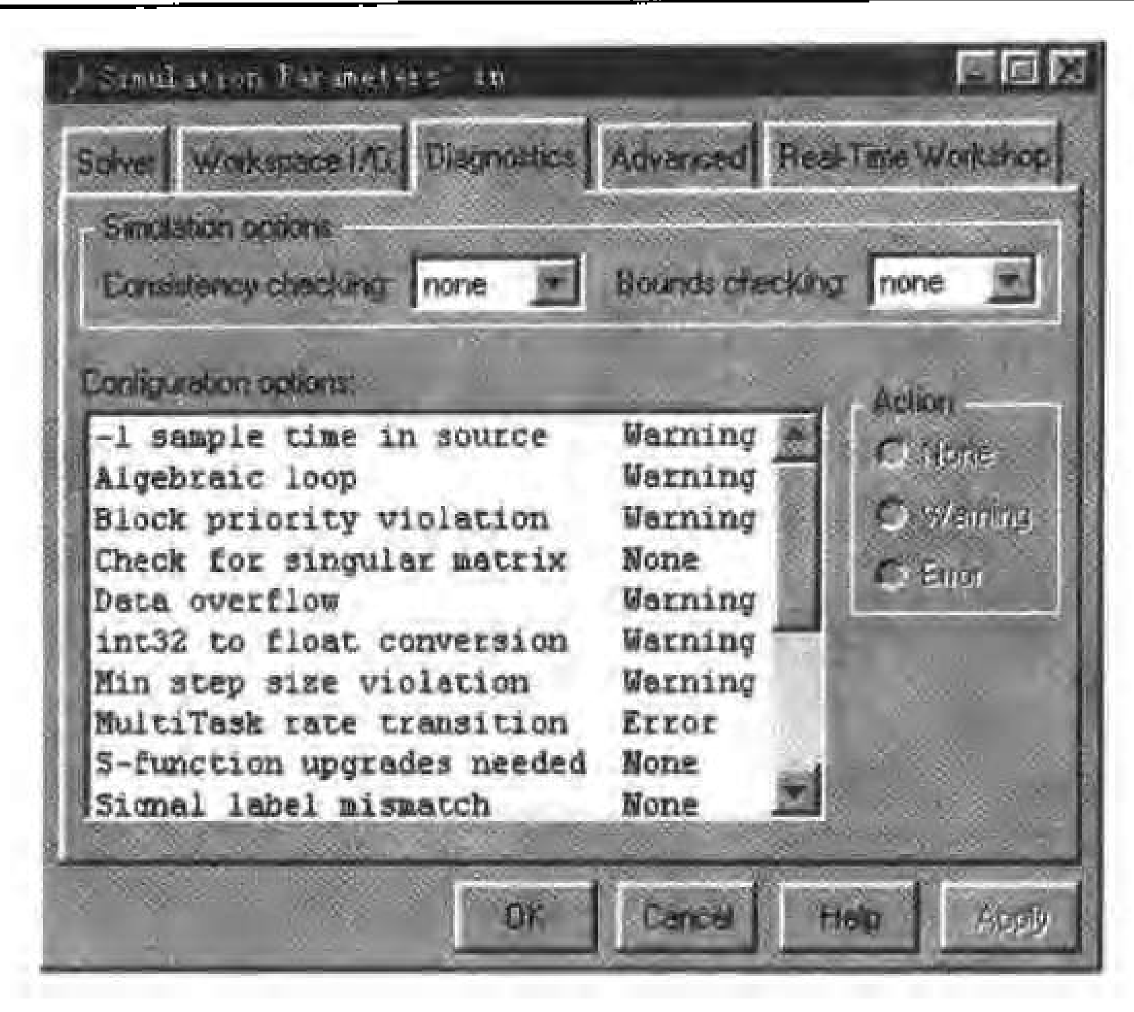


图 7－9 Diagnostics 页(诊断页)

里选择相应的处理方式。

表 7－1 列出了列表框中列举的一些不正常事件的意义。

表 7－1 不正常事件的意义

事 件	说 明
－1 smaple time in source	源模块设定采样时间为－1
Algebraic loop	Simulink 运行仿真时,检测到代数环
Check for singular matrix	Product 模块在矩阵乘法模式转化其中的一个输入时,检测到奇异矩阵
Data overflow	信号或者参数的值太大,以致它们的数据类型无法表示
Int32 to float conversion	32 位整数被转化为浮点数,这种转换会产生精度的损失
Min step size violation	下一个仿真步小于模型所设定的最小仿真步,这可能发生在设定的误差限要求比最小步长要小的仿真步
Multitask rate transition	在运行在多速率模式下的两个模块间进行无效的速率转换
S－function upgrades needed	一个没有使用当前版本 S－函数特性的模块被遇到
Signal label mismatch	仿真遇到几个具有相同的源信号但不同的标签名的虚拟信号
Single Task rater ransition	速率转换发生在运行在不同速率下的两个模块间
Unconnected block input	模型包含具有没有连接的输入端的模块
Unconnected block output	模型包含具有未连接输出端的模块
Unconnected line	模型包含为连接的直线
Unneeded type conversion	一个类型转换模块出现在类型转换不是必需的地方
Vector/Matrix conversion	向量到矩阵的转换或者矩阵到向量的转换发生在模块的输入端
Block Priority Violation	当运行模型时,Simulink 发现模块优先级设置错误

而 Options 而板里的几个检查框的意义时:

1. Consistency Checking 选项

一致性检验的是一种验证 Simulink ODE 解法器所作的几条假设的调试工具。它的主要作用是使 S-function 和 Simulink 以同种方式工作，但是这种检查会导致仿真效率的急剧下降(最多可达 40%)，因此，一般来说它总是被设为 off 的。但在使用 S-function 时，一致性检验可以帮助用户判定产生错误的原因。一致性的另外一个目的，是使模块在某个时间，被调用时获得常数的输出，这在解决 stiff 问题非常有用。

2. Disabling Zero Crossing Detection 选项

选中这个选项禁止仿真时进行过零检测，对于一个具有过零点的模型，这样可以加快仿真的速度，但是会影响仿真结果的精度。一般它只用于那些本身具有内部的过零检测的模块，它不能禁止 hit crossing 模块的过零检测。

3. Disable optimized I/O storage

选中这个选项使 Simulink 为模块的每一个 I/O 值分配一个独立的缓存，而不是复用同一个缓存。这样在仿真大的模型时，会大大增加所需的存储空间。所以最好是在调试时才选中这个选项。特别是在以下几种情况，读者要禁止缓存的复用：

(1)调试一个 C-MEX 格式的 S-函数；

(2)使用一个浮动 Scope 或者 Display 来监视所调用模型的信号。如果在使用一个缓存已被复用的浮动 Scope 或者 Display 时，缓存复用依然可以使用，那么 Simulink 就打开一个错误对话框。

4. Relax boolean type checking 检查框

它的作用主要是为了和以前的 Simulink 版本相兼容。这样对于一个只能输入布尔数据类型的模块能输入 double 类型的数据。这一点很好理解。请看下面的示例模型。如图 7-10所示。

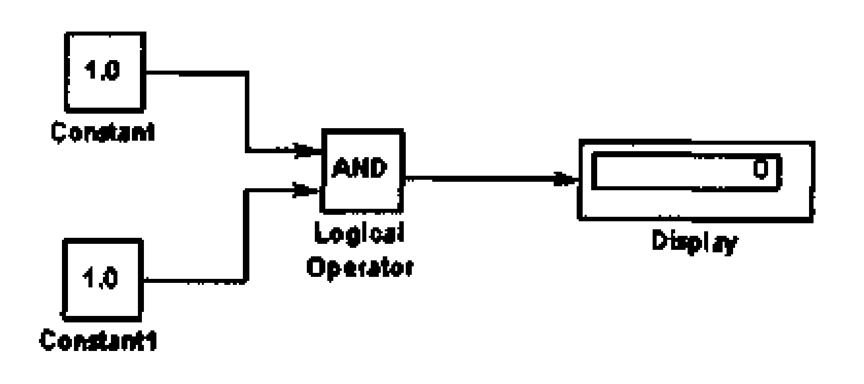


图 7-10 布尔类型检查示例模型

读者可以改变这个检查框的选中与否，来试试运行仿真后的结果。读者可以发现在选中检查框时，模型可以没有错误的运行，但是一去掉就会出现错误。

上面所述的 4 个选项在 Simulink3.0 都是放在诊断页的，但是在 Simulink4.0，有一部分则放在了 Advanced 页。

7.2.4 Advanced 页

Advanced 页是 Simulink4.0 新增加的页，它上面列置了原来放在诊断页的一些选项设置。图 7-11 显示了它的外貌。

这个页分为两个面板：一个是模型参数配置面板，一个则是优化面板。

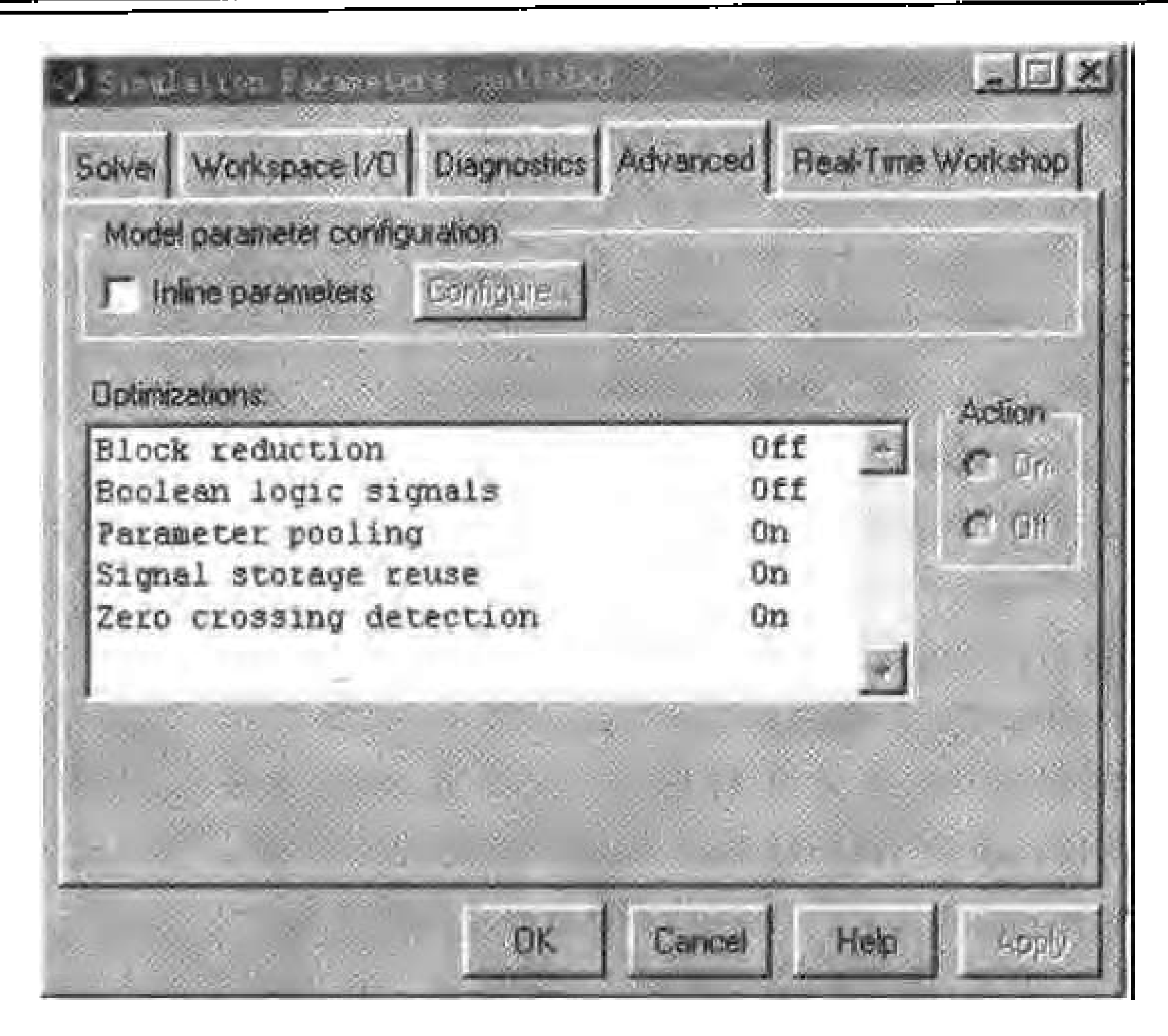


图 7-11 Advanced 页

在模型参数面板，读者可以选中 Inline parameters 检查框。这样将使所有的参数变成不可调谐的(在仿真运行时变化)，除了那些用户特别设定的以外。将参数设为不可调谐，会使 Simulink 把它们当成常数，这样就可以加速仿真的运行。使用 Configuration dialog box 对话框可以设定在 Inline parameters(内嵌参数)选中时，用户想保留为可调谐的变量。这个对话框可以通过检查框旁边的 configue 按钮来打开。图 7-12 是它的示意图。

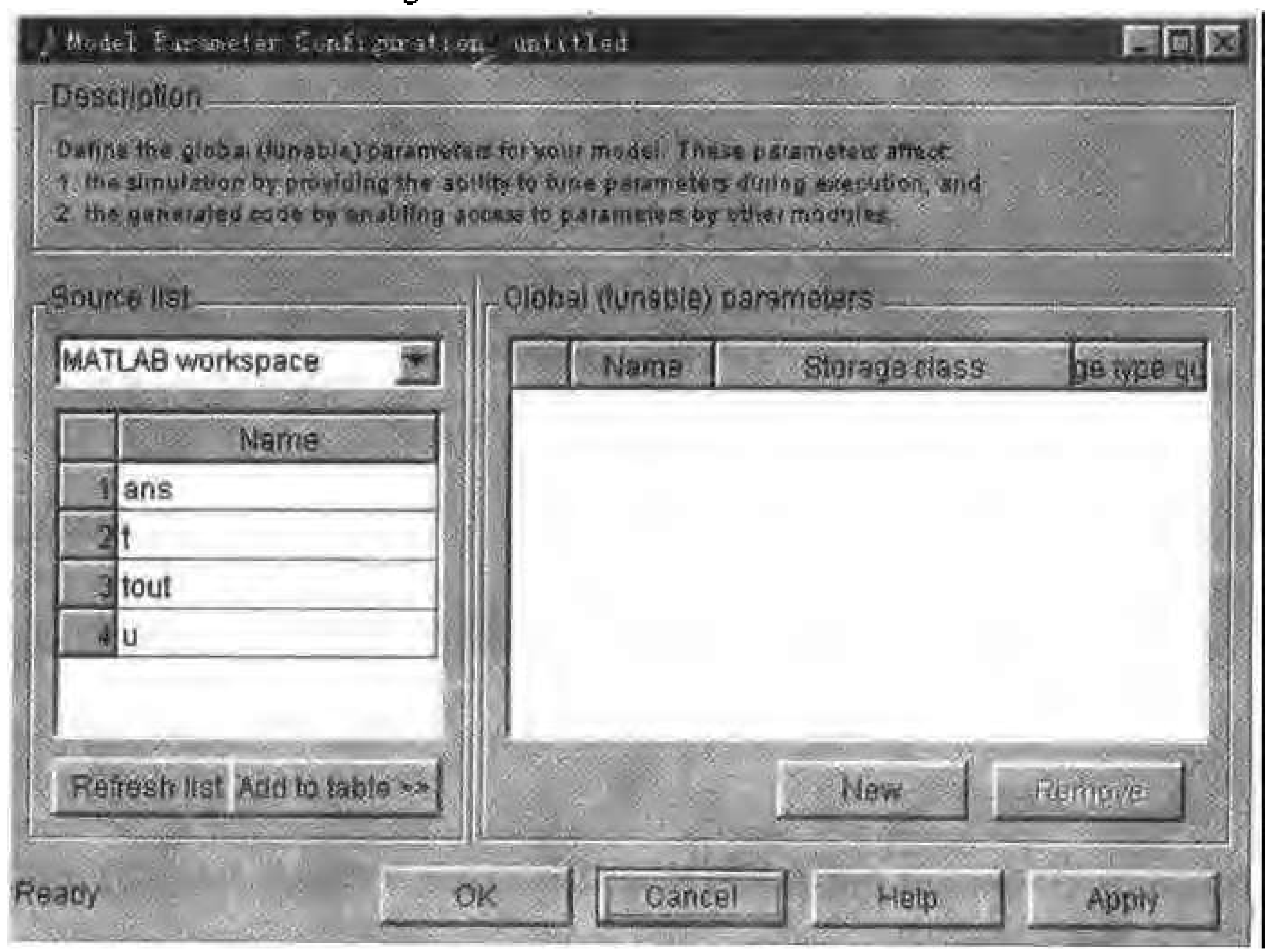


图 7-12 参数配置对话框

这个对话框的使用是十分简单的，读者只需在对话框左边源列表里的变量中，选择虚设定为可调谐的，用左下角的 Add to table 按钮，就可以添加到右边的全局列表里，也就变成了可调谐的(全局的)。至于列表中的 Storage Class 和 Storage type qualifier 两个属性，都是用于代码生成的。

当这个选项被选中时，只有符合下列条件的参数才能在仿真过程中变化：

(1)参数的值必须是 MATLAB 工作空间的变量；

(2)模型参数必须是在模型参数配置对话框设定为 global 的变量。

改变符合上述条件的参数，可以通过改变相应的工作空间的变量值，并且用 Edit 菜单下的 Update Diagram 命令来更新图表。

而在优化面板，有一些选项在 Simulink3.0 中是位于诊断页的，如 Boolean logic signals, Signal storage reuse 和 Zero - crossing detection，关于它们的解释请看前面一小节。除此之外的两个选项的意义如下：

(1)Block reduction。用一个合成的模块来代替一组模块，提高仿真速度；

(2)Parameter pooling。这个选项用于代码生成，在不进行代码生成时，这个选项可以不去考虑。

7.3　改善仿真的性能和精确度

仿真的性能和精确度受很多因素的影响，包括模型的设计和仿真参数的选择。对于大多数问题，使用缺省的仿真参数值、解法器可以精确而有效地解决。但是有些模型，适当地调整解法器和仿真参数，可以得到更好的仿真结果。并且，如果用户知道模型行为的选项，并把它告诉解法器，也可提高性能。

7.3.1　加速仿真

一个模型的仿真速度过慢，是由许多因素造成的。下面是其中的一些因素：

(1)模型包括一个 MATLAB Fcn 模块。当执行一个包含 MATLAB Fcn 模块的模型时，Simulink 在每一个仿真时间步都要调用 MATLAB 解释器。所以应该尽可能地使用 Simulink 的内置 Fcn 模块或者是最基本的 math 模块。

(2)模型包含用 M 文件的 S - 函数，M 文件 S 函数同样会使 Simulink 在每一个仿真时间步调用 MATLAB 解释器。代替的方法是把 M 文件的 S - 函数转化成 c - mex 函数或者是建立一个等价的子系统。

(3)模型包含一个存储模块。使用存储模块将使阶数可变的解法器(如 ode15s 和 ode13)在每个仿真时间步被重置回 1 阶。

(4)仿真的时间步长太小。解决的方法是把最大仿真步长参数设置为 Simulink 的缺省置——auto，看看效果如何。

(5)仿真的精度要求过高。一般来说，相对误差限设为 0.1%就已经足够。当模型存在取值会趋向于零的状态，仿真时如果绝对误差限度太小，会使仿真在接近零的状态附近耗费过多的仿真步。

(6)仿真的时间过长。可酌情减小仿真的时间的间隔。

(7)所解决的问题是 stiff 问题,去选择了一个非 stiff 的解法器,可以试试 ode15s。

(8)模型所设置的采样时间的公约数过小,这样使 Simulink 可采用的基准采样时间过小,因为 Simulink 会选择足够小的时间步确保所设置的采样点都能取到。

(9)模型包含一个代数环。代数环的求解方法就是在每一个时间步迭代地进行计算,这当然会严重地降低仿真的性能。

(10)模型把一个 random number 模块作为 integrator 模块的输入。对于连续系统,可以使用 source 子库里的 Band - Limited WhiteNoise(带限白噪声)模块。

7.3.2 改善仿真的精度

检验仿真精度的方法是,修改仿真的相对误差限和绝对误差限,在一个合适的时间跨度反复运行仿真,看看仿真的结果有没有大的变化,如果变化不多,则表示解是收敛的。

如果仿真在开始时错过了模型的关键行为,那么可以更改初始步长使仿真不会忽略这些关键的行为。

如果仿真的结果不稳定,可能是以下原因所致:

(1)系统本身不稳定;

(2)如果正在使用 doe15s,用户可以把最大阶数定为 2 或者尝试 ode23s;

(3)如果仿真的结果看起来不是很精确,它的原因可能使:

- 模型有取值接近零的状态,如果模型的绝对误差限过大,会使仿真在接近零的区域运行的仿真时间步太少。解决的办法是修改绝对误差参数或者在积分模块的对话框修改初始的状态;
- 如果改变绝对误差限还是不能达到预期的误差限,请修改相对误差限,使可接受的误差降低,并减小仿真的步长。

7.4 从命令运行仿真

相对于使用菜单命令,从命令行运行仿真使得用户可以从 M 文件来运行仿真,这样就允许仿真和模块参数可以被不断地改变参数。也就可以让用户随机地改变参数和循环地运行仿真,就可以进行蒙特卡罗分析了。有两个命令可供用户选用:sim 和 set _ param 命令。

7.4.1 使用 sim 命令

sim 命令的作用是仿真一个由 Simulink 模型表示的系统,它的完整格式是:

```
[ t, x, y, ] = sim (model, timespan, options, ut) ;
```

或者

```
[ t, x, y1, y2,…, yn ] = sim (model, timespan, options, ut ) ;
```

其中,只有 model 参数是要求的,右边其他的参量,都被允许置为空矩阵([])。命令中设置过的参数将会覆盖在模型中定义的参数值(如用仿真参数对话框等)。当仿真的模型是连续系统,那么命令中还必须设定 solver 参数,这可以使用 simset 命令。下面是各个参量的详细说明。

t 返回仿真的时间向量。

x 返回仿真的状态矩阵，排列顺序是先连续状态，然后是离散状态。

y 返回仿真的输出矩阵，其中的每一列对应着一个根层次的输出端口（即顶层系统），顺序按端口数字对应。如果一个输出端口的结果是向量信号，那它相应地占有合适的列数。

y1，……yn 返回模型中的根层次输出端口的输出，这样的端口模型有 n 个。

model 一个模块图表的名称。

timespan 仿真的起始和终止时间。有两种说明方式：tFinal 仅指定结束时间，而起始时间为 0；[tStart tFinal] 说明开始和结束时间。

options 由 simset 命令建立的结构，用于指定可选的仿真参数。

ut 可选的对顶层输入端口模块的外部输入。Ut 可以是一个 MATLAB 函数（用 string 表达），指定每一个仿真时间步的输入 u = UT(t)，UT 表示输入和 t 的关系，或者是一个用逗号分割的列表，ut1，ut2，…，其中的每一个对应着一个输入端口。面向所有端口表格化输入可以是 MATLAB 矩阵或者结构的形式。面向单个端口的表格化输入只能是结构的形式。

下面是它的一个使用示例。它的仿真 Van der Pol 方程模型，这个命令全部使用缺省参数：

```
>>[ t, x, t ] = sim ('vdp')          %vdp 是 Simulink 的演示模型
```

面下面的命令，使用了 vdp 模型图标上的参数，但是定义了 Refine 参数的值：

```
>>[t, x, y ] = sim ('vdp', [], simset ('Refine', 2 ) ;
```

第三格命令仿真 vdp 模型 1000 s，并保存返回变量的最后 100 行，仿真的输出变量仅包含 t 和 y，但是在 xFinal 保存最终状态向量。

```
>>[t, x, y] =sim ('vdp', 1000, simeset('MaxRows', 1000, 'OutputVariables', '
ty', 'FinalStateName', 'xFinal') ) ;
```

7.4.2 使用 set _ param 命令

可以使用 set _ param 命令来开始、结束、暂停或者继续仿真，或者更新模块图表。类似地，还可以使用 get _ param 命令来检查一个仿真的状态。set _ param 命令的使用格式是：

set _ param ('sys', 'SimulationCommand', 'cmd')

其中，'sys'是系统的名称，'cmd'是控制命令的取值，有：'start'，'stop'，'pause'，'continue'，或者'update'。

get _ param 命令的使用格式是：

get _ param('sys', 'SimulationStatus')

这个命令的返回值可以为：'stopped'，'initializing'，'running'，''paused'，'terminating'和'external'。

7.5 分析仿真结果

7.5.1 观看输出结果的轨迹

Simulink 中，观看仿真的输出轨迹，有三种方法：

(1)把信号输入到 Scope 模块或者 XY Graph 模块;

(2)把输出写入返回变量,并用 MATLAB 命令绘图;

(3)使用 To Workspace 把输出写入到工作空间,让后用 MATLAB 命令绘制出仿真的图。

1. 使用图形显示模块

第一种方法的 Scope 模块在前面已经多次使用,这里就不多讲了,只谈一下 XY Graph 模块和 Scope 的区别。Scope 模块,作出的图是以时间为横坐标,以输出为纵坐标。而对 XY Graph,则可以绘制出一个信号相对于另一个信号的图形。

图 7 - 13 中的 Cos Wave 模块是将 Sinewave 的相位设为 pi/2 得到,这样 XY Graph 模块显示的图形位如图 7 - 14 所示。

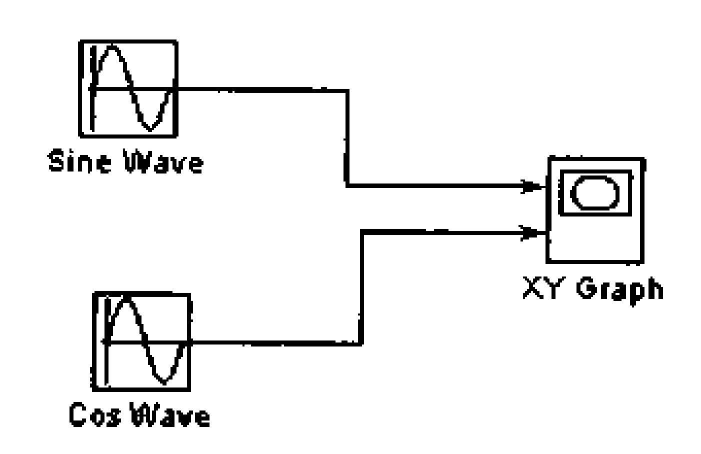


图 7 - 13　XY Graph 模块使用

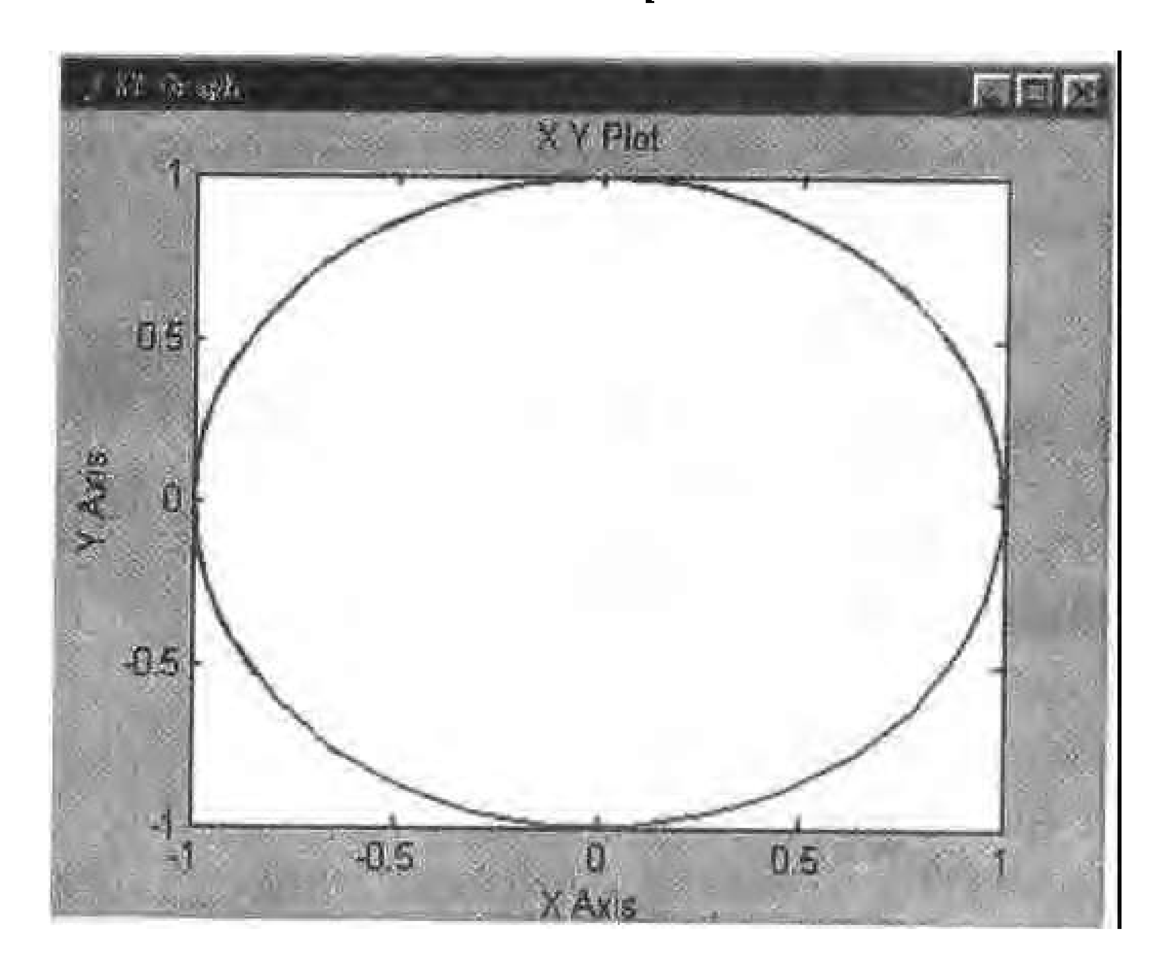


图 7 - 14　XY Graph 模块显示的图形

上述图形在判断信号是否同步时有很大的用处。

2. 使用返回变量

使用返回变量的首要步骤是在设置参数对话框里,选中输出检查框 time 和 output,再把相应的时间变量和保存输出的变量设置好,一般而言,它们缺省值被选中,而且变量名分别为 tout 和 yout。下面来看一个例子,首先请按图 7 - 15 建好示例模型。

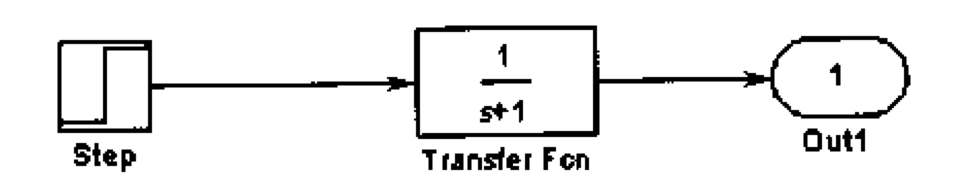


图 7-15 示例模型

其中 Step 模块在 source 子库，Transfer Fcn 模块在 continous 子库，模型中的输出端口是必需的，它表示对应的信号就是上面的输出信号。运行仿真后，读者会发现 MATLAB 工作空间多了两个变量：tout 和 yout，这就是由 Simulink 保存在 MATLAB 工作空间的返回变量。如图 7-16 所示。用 plot 命令就可以绘出所需的图形。

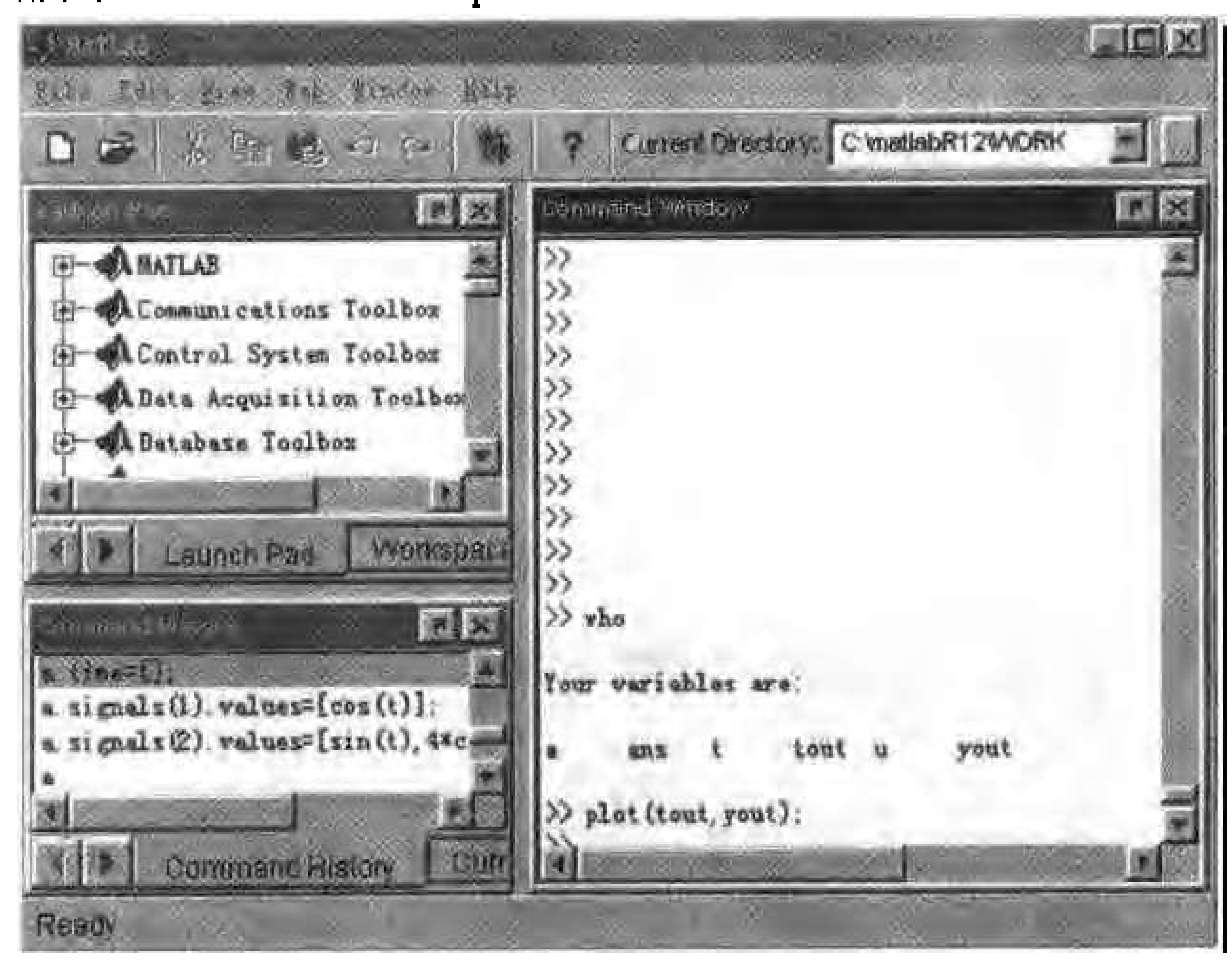


图 7-16 命令窗口的情况

```
> >plot (tout, yout);
```

运行结果如图 7-17 所示。

3. 使用 To Workspace 模块

最后一种方法是使用 To Workspace 模块，它在 sinks 子库，还是用上面的例子。请读者把 out1 模块用 To Workspace 模块代替。如图 7-18 所示。

To Workspace 模块的参数对话框如图 7-19 所示。

图上标出使用 To Workspace 模块时两个比较重要的参数。请运行仿真

```
> >who
Your variables are:
simout    tout
> >simout
simout =
```

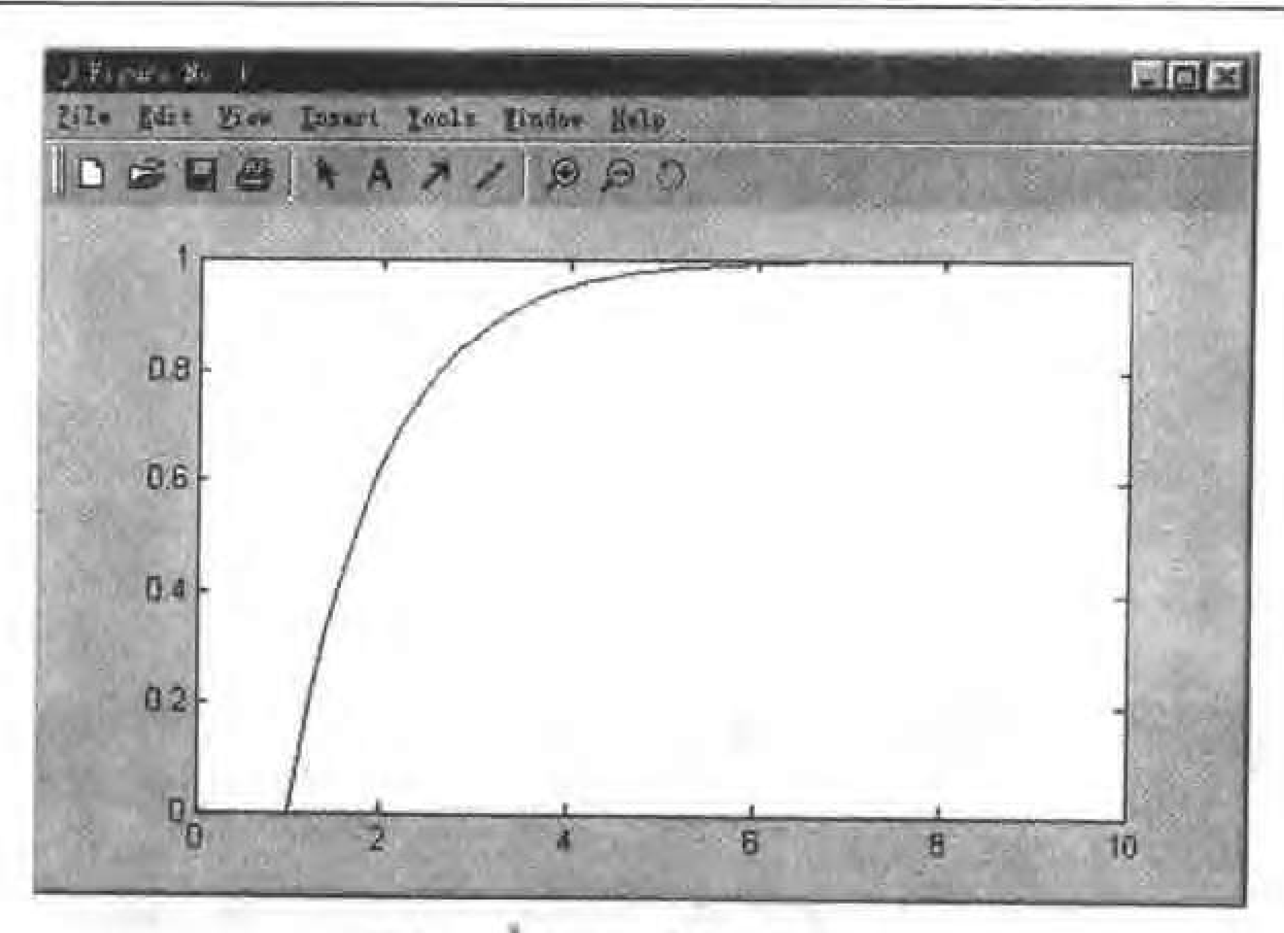


图 7-17 用 plot 命令得到的图形



图 7-18 替换后的模型

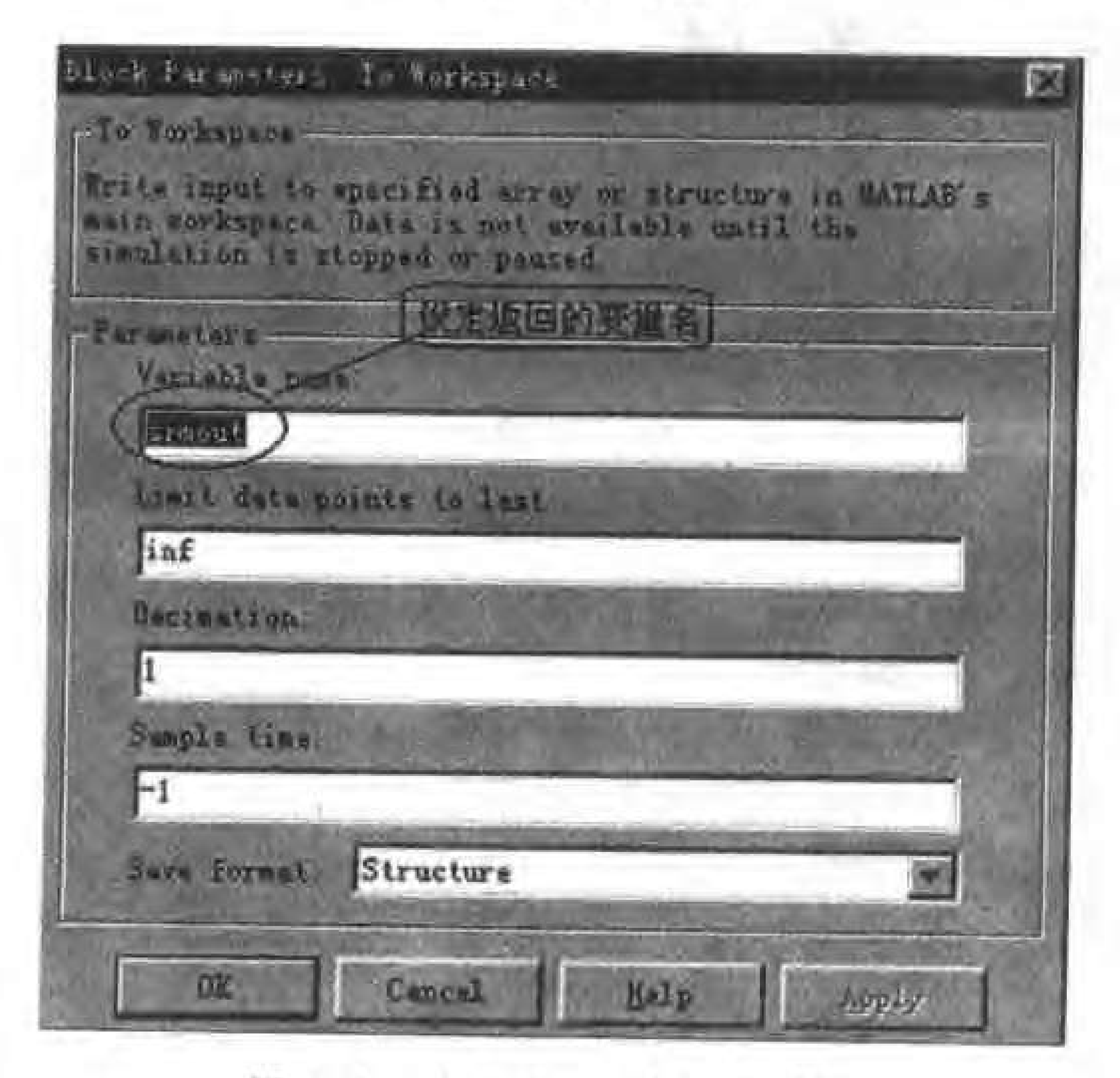


图 7-19 To Workspace 的参数对话框

```
        time : []
        signals : [|x| struct]
        blockName : 'testreturnout/To Workspace'
>>plot (tout, simout, signals.values) ;
```

7.5.2 线性化

Simulink 提供了 linmod 和 dinmod 连续函数来抽取用状态空间矩阵形式 A、B、C 和 D 表示的线性模型。状态空间矩阵按下面的方程来描述线性的输入和输出关系：

$$x = Ax + Bu$$

$$y = Cx + Du$$

其中，x，u，y 分别表示状态、输入和输出向量。例如，图 7-20 是称为 lmod 的模型，其中 Plant 模块是把 Transfer Fcn 模块的 denominator 参数设为 [1 2 1]得到的。

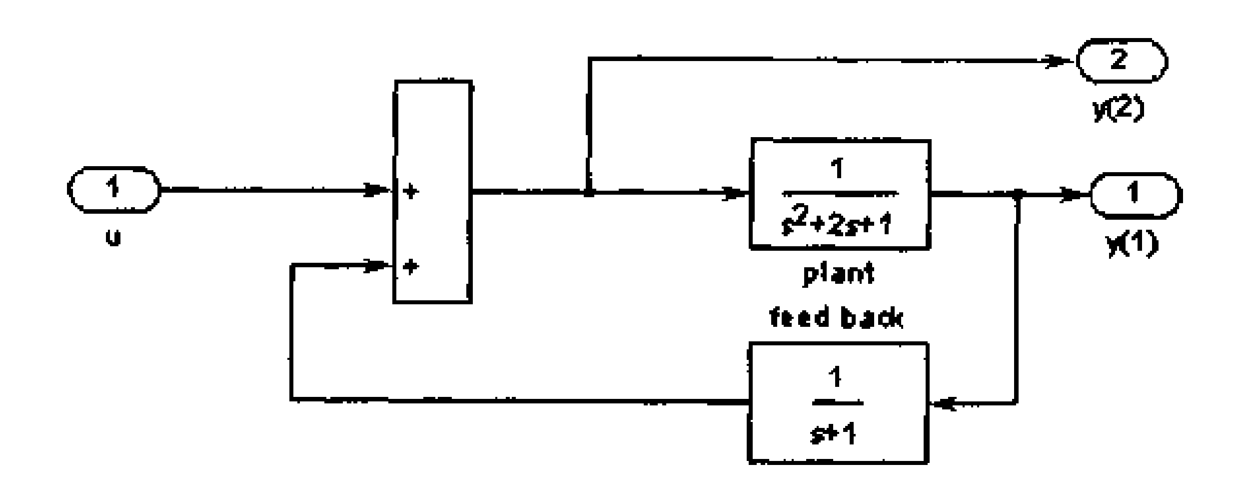


图 7-20　lmod 模型图表

保存模型后，用下面的函数就可以把模型变成线性状态方程的形式。

```
>>[A, B, C, D] = linmode ('lmod') %lmod是模型的名称
A =
    -2    -1     1
     1     0     0
     0     1    -1
B =
     1
     0
     0
C =
     0     1     0
     0     0     1
D =
     0
     1
```

一旦数据具有状态空间的形式，或者被转化为 LTI 对象，用户可以使用控制系统工

具箱里的函数进行进一步的分析。

• 将状态空间转化为 LTI 对象,使用

sys = ss (A, B, C, D)

得到

```
a =
                  x1          x2          x3
      x1          -2          -1           1
      x2           1           0           0
      x3           0           1          -1

b =
                  u1
      x1           1
      x2           0
      x3           0

c =
                  x1          x2          x3
      y1           0           1           0
      y2           0           0           1

d =
                u1
      y1         0
      y2         1
Continuous - time model.
```

• 绘制波特相位幅频图,使用

bode (A, B, C, D)或者 bode (sys)

得到如图 7 - 21 所示曲线图。

• 求出线性化时间响应,可以使用的命令有:

step (A,B, C, D)或者 step (sys),求单位阶跃响应,结果如图 7 - 22 所示;

impulse (A, B, C, D)或者 impulse (sys),求系统的单位冲击响应,结果如图 7 - 23 所示。

下面举一个例子来说明如何对一个系统进行线性化,并据此求出它的时间响应曲线。首先使用 linmod 对 f - 14 进行线性化:

```
> >[ A, B, C, D] = linmod ('f14)
> > t = 0 : 1: 10 ;
> > y = step (A, B, C, D, 1, t ) ;
> >plot (t,y) ;
```

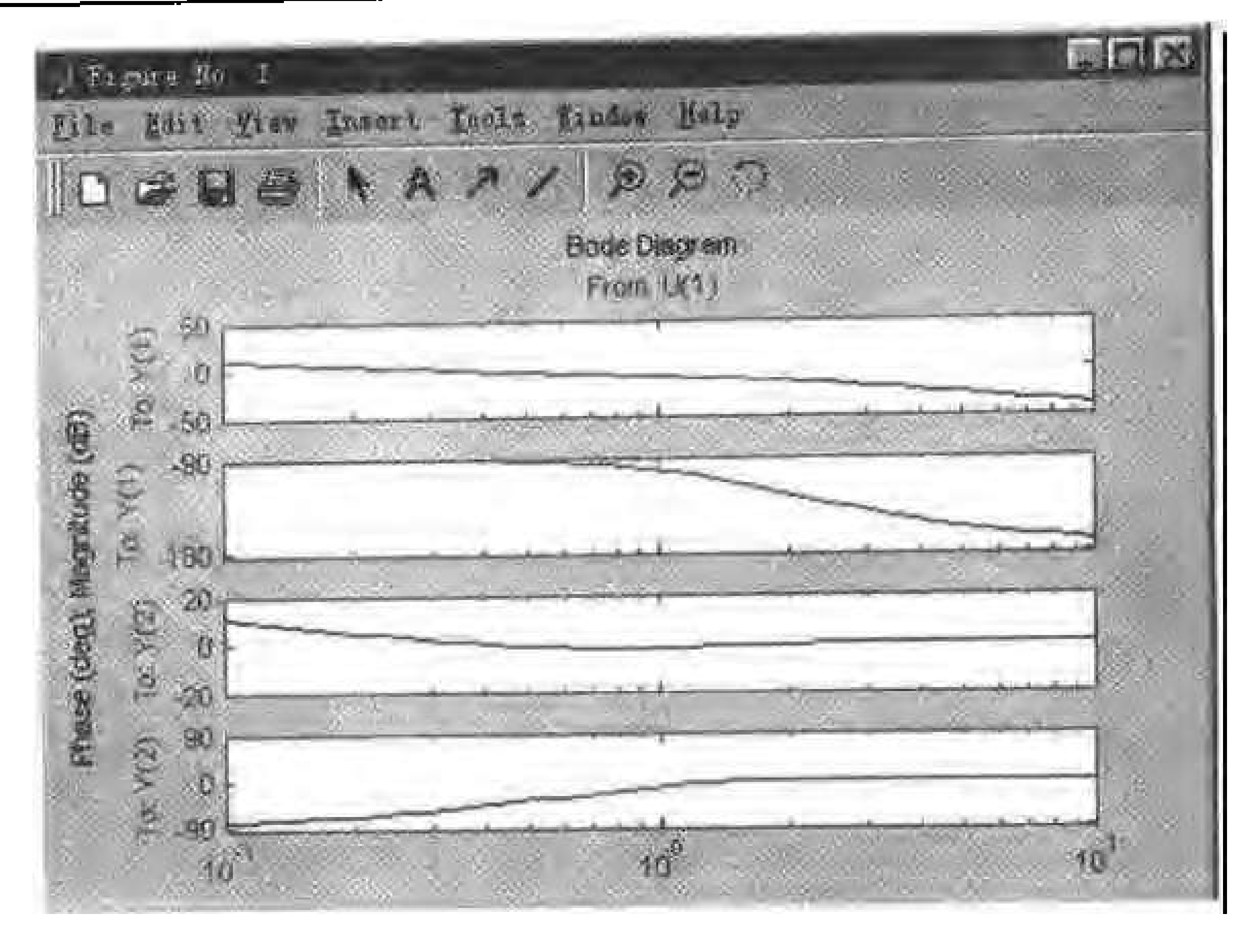


图 7-21 绘制波特相位幅频图

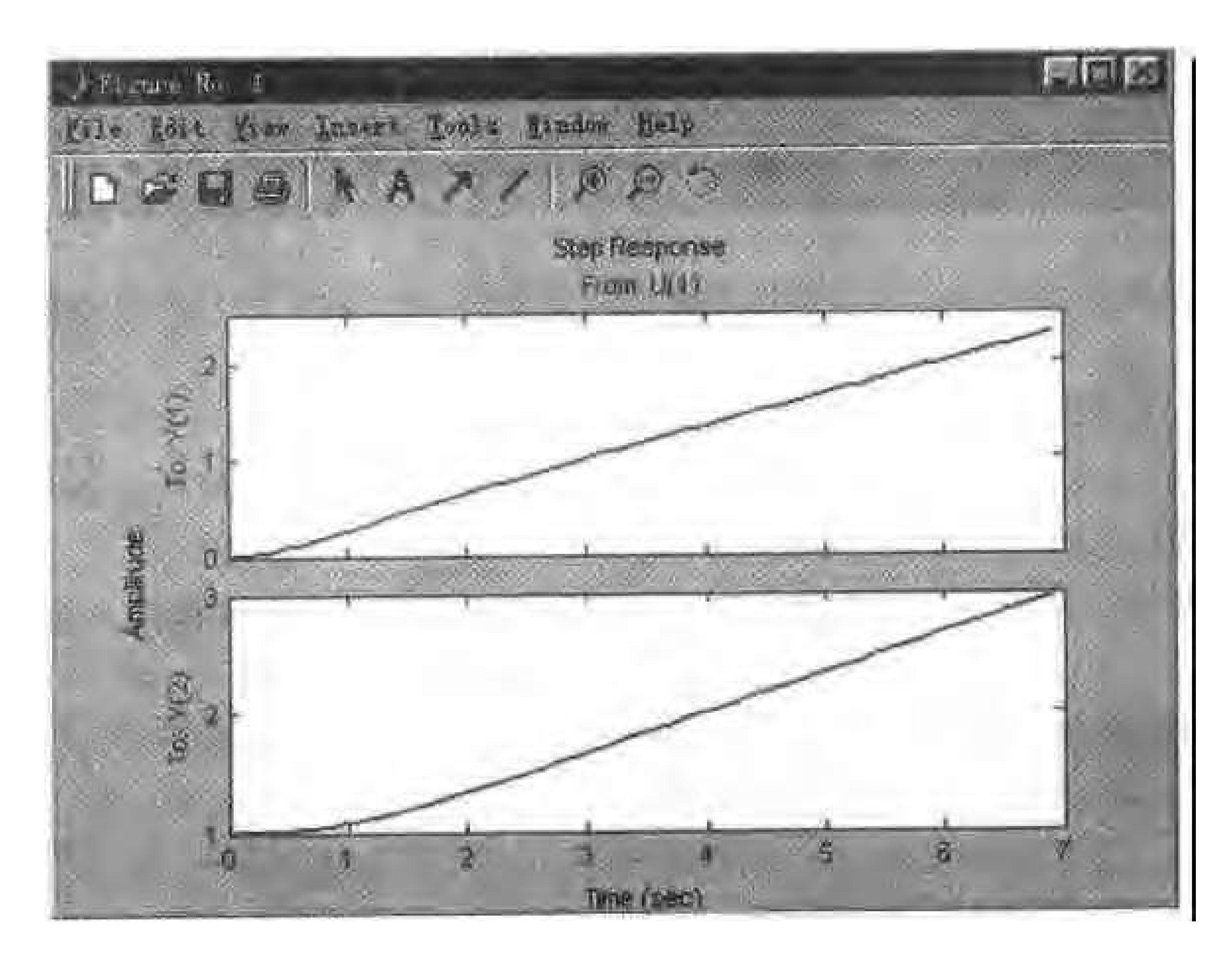


图 7-22 单位阶跃响应图

这样,就可以绘出通过对 f-14 的线性化系统对 f-14 进行线性化仿真,在上面的命令中,使用 step 求出了系统的阶跃输出响应。

如果模型本身是非线性的,那么要选取一个工作点,用来标识从哪里抽取线性化的模型。非线性模型同样对摄动的大小很敏感,所以必须存在一个截断误差和舍入误差的折中。Linmod 函数提供了额外的参数来说明工作点和扰动的大小。使用的命令为:

[A, B, C, D] = linmod ('sys', x, u, pert, xpert, upert)

对于离散系统或者离散和连续的混合系统进行线性化,就要使用 dlinmod 函数,它的语法和 linmod 基本类似,只是使用时必须包括一个说明采样时间的参数。

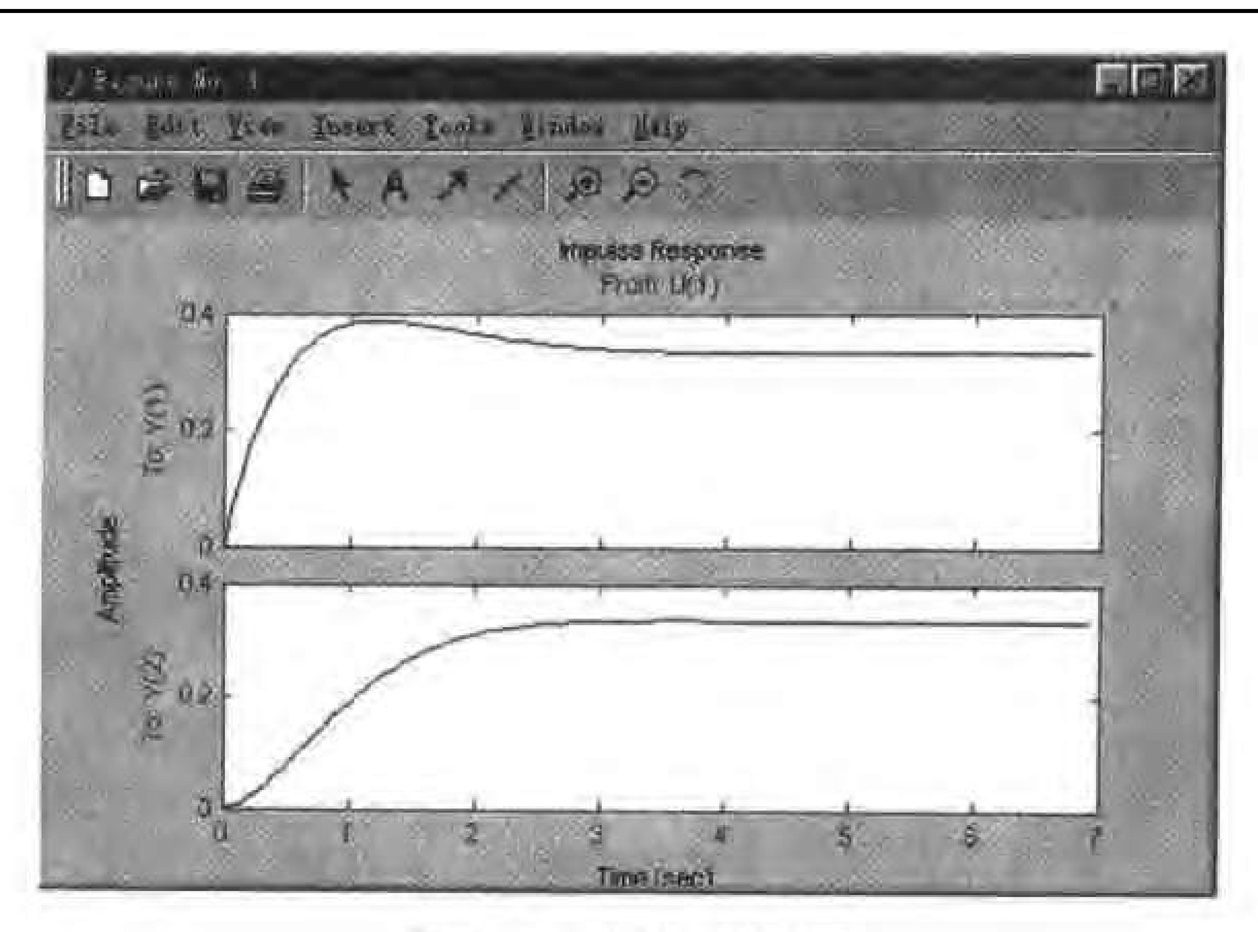


图 7-23 单位冲击响应图

[A, B, C, D] = dlinmod ('SYS', TS)

更详细的信息请看 dlinmod 的帮助信息。

用 Linmod 函数对包含微分模块或者传送时延模块的模型进行线性化，将会出现问题。因此，在进行线性化之前，必先用经过特殊设计的能避免这个问题的模块代替上面的微分模块和传送时延模块。这些替代模块的位置是在 Simulink Extras 的 Linearization 子库，它们是：

(1)对于 Derivative 模块，请用 Switched derivative 模块来替代。

(2)对于 Transport Delay 模块，请用 Switched transport delay 模块来替代(这个模块只有安装了控制系统工具箱，才能被使用)。

对于微分模块的另外一种处理方式是，尽量把微分项合并到其他的模块中。这句话的意思是，例如，对于一个微分模块后串接一些 Fcn 模块的实现形式，用一个具有传递函数 sl(s+a)的 Fcn 模块的形式来实现会更好一些(尽管这并不总是可以办到的)。与图 7-24所示的一样，右边的这种串接形式用左图单个 Fcn 模块实现会更好些。

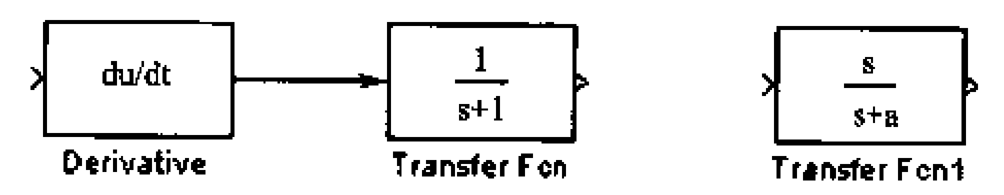


图 7-24 用传递函数来替换微分模块

2. 线性化函数使用方法

linfun 支持的调用格式有：

[A, B, C, D] = linfun ('sys')

[A, B, C, D] = linfun ('sys', x, u)

[A, B, C, D] = linfun ('sys', x, u, pert)

[A, B, C, D] = linfun ('sys', x, u, pert, xpert, upert)

其中各个参数的意义分别是：

linfun　线性化函数，包括：linmode，dlinmod，linmod2。

sys　需要线性化的系统名称。

x 和 u　状态和输入向量，如果定义这些量，那么这些量就设置了工作点，在这个点上提取线性模型。

pert　可选标量，用来设置 x 和 u 的干扰。如果这个值没有定义，就采用默认的值 1e－5。

xpert 和 upert　可选向量，用来为每一个状态和输入设置扰动。如果定义了这个向量，那么就忽视双面上的标量 pert，这种情况下，状态 x 和第 i 个元素收到的扰动后就变成 x(i)＋xpet(i)；输入 u 的第 j 个元素收到的扰动后就变成 u(j)＋upert(j)。

现对它的使用说明如下：

linmod 得到的是用常微分方程描述的 Simulink 模型的线性模块。返回的模块用状态空间 A、B、C、D 形式来表示其输入和输出的关系，这样的线性模型可用下面的式子来表示：

$$x = Ax + Bu$$
$$y = Cx + Du$$

[A, B, C, D] = linmod ('sys')可以得到在状态变量 $x=0$ 和输入 $u=0$ 这个工作点附近的线性模型。Linmod 是在工作点附近对状态施加扰动后来确定状态，确定状态导数和输出的变化速率(即雅可比矩阵)，并把所得的结果用来计算状态空间矩阵。每一个状态受到扰动后变成

$$x(i)+\Delta(i)$$

其中，

$$\Delta(i)=\delta(1+|x(i)|)$$

同样，第 j 个输入受到扰动后，变成

$$u(j)+\Delta(j)$$

其中，

$$\Delta(j)=\delta(1+|u(j)|)$$

3. 离散系数的线性化

dlinmod 能够以任意给定的采样时间对离散系统、多速率系统，以及连续和离散这类混合系统进行线性化。除了要求在第二个选项输入采样时间来对系统线性化外，dlinmod 的调用格式和 limod 基本相同。例如：

[Ad, Bd, Cd, Dd,] = dlinmod ('sys', Ts, x, u) ;

上面这个命令可以产生一个以状态向量 x 和输入向量 u 为工作点、采样时间为 Ts 的离散状态空间模型。要想得到一个离散系统的近似连续模型，只要把 Ts 设为 0 即可。如果一个模型是由线性模块、多速率模块、离散模块和连续模块组成的，那么在同时满足下列条件的情况下，dlinmod 产生一个采样时间为 Ts、且具有相同频率和时间响应的线性模型：

(1) Ts 是系统中所有采样时间的整数倍；

(2)Ts 是不小于系统中最慢的采样时间；

(3)系统是稳定的。

不过，在上面这些条件不满足的情况下，也有可能得到有效的线性模块。要看一个系统是否稳定，实际上只要计算线性化后所得到的矩阵 Ad 的特征值就可以了。因此，如果 Ts>0 而且特征值在单位圆里，即

```
all (abs (eig (Ad))) < 1      %eig 表示求矩阵 Ad 的特征值
```

那么系统是稳定的。同样，如果 Ts = 0，而且所有的特征值在左半平面，那么系统也是稳定的。即

```
all (real (eig(Ad) )) < 0
```

当系统不稳定且采样时间不是其他采样时间的整数时，dlinmod 就可能产生复矩阵 Ad 和 Bd。然而在这种情况下，仍然可以通过矩阵 Ad 的特征值来验证系统的稳定性。

7.5.3 平衡点的分析

1. 确定平衡点

所谓平衡点，也称为 trim 点，就是指参数空间中使动态系统处于稳定状态的点，在数学上，平衡点就是使状态的导数为 0 的工作点，既包括输入值 u 和状态值 x。例如，飞行中的飞机的平衡点就是使飞机竖直和水平飞行的控制设置。

确定系统的稳定平衡点的函数是 trim，还是以上面的 lmod 模型，为例来演示如何确定平衡点。如图 7-25 所示。例如，用 trim 来寻找使两个输出都为 1 的输入和状态值。

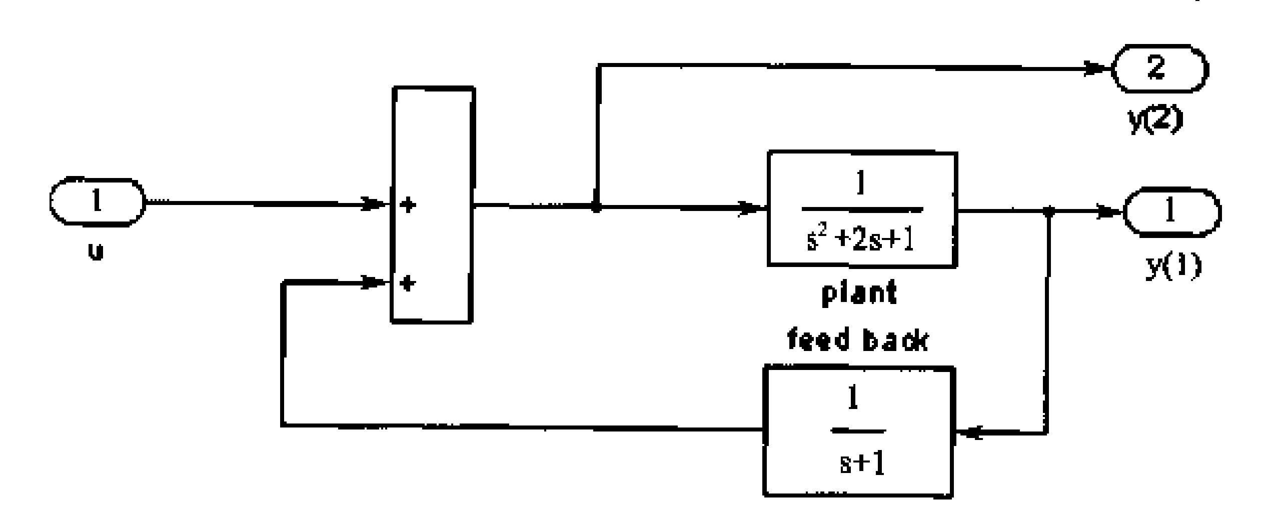


图 7-25 lmod 模型

首先，对状态和输入值进行一个初步的猜测，并把预期的输出值赋给 y。

```
>>x = [0; 0; 0];
>>u = 0 ;
>> y = [ 1; 1];
```

然后，要用索引变量来规定模型的输入、输出和状态中哪些可以变化，哪些不能变化。

```
>> ix = [] ;        %不固定状态变量中任何一个
>> iu = [] ;        %不固定输入
>> iy = [1; 2] ;    %固定输入端口 1 和输出端口 2
```

调用 trim 命令来求出稳定点

```
>> [x, u, y, dx] = trim ('lmod', x, u, y, ix, iu, iy)
```

```
x =
     0.0000
     1.0000
     1.0000
u =
     3.0537e-016
y =
     1
     1
dx =
     1.0e-015 *
              0
         0.0402
         0.2220
```

2. trim 命令使用格式

下面就来详细介绍 trim 函数的用法，trim 函数支持的格式有：

```
[x, u, y, dx] = trim ('sys')
[x, u, y, dx] = trim ('sys', x0, u0, y0)
[x, u, y, dx] = trim ('sys', x0, u0, y0, ix, yi)
[x, u, y, dx] = trim ('sys', x0, u0, y0, ix, yi, dx0, idx)
[x, u, y, dx] = trim ('sys', x0, u0, y0, ix, yi, dx0, idx, options)
[x, u, y, dx] = trim ('sys', x0, u0, y0, ix, yi, dx0, idx, options, t)
[x, u, y, dx] = trim ('sys',…)
```

我们知道，trim 命令就是获得使状态的微分为零的输入 u 和状态 x 的值。trim 命令的工作原理大致是，从一个初始值开始（调用时，需要先指定），然后使用二分查找的算法来搜索，直至找到最接近的 trim 点。当 trim 命令找不到平衡点，它就会返回在搜索中遇到的使状态微分在 min - max 意义下最接近零的点。即是使距微分的零点的最大偏离最小的点。trim 命令可以找出符合特定输入、输出状态条件的 trim 点，和让系统按指定方式变化的点。也就是，使系统的状态微分等于特定的非零值。

[x, u, y] = trim ('sys') 找到距系统初始状态 x0 最近的平衡点，特别是 trim 找出使 [x - x0, u, y]的最大绝对值最小的平衡点。

如果找不到这样的平衡点，它将返回使 abs(dx)最小的点。这时就可以使用下面的命令来获得 x0 的值：

```
> >[sizes, x0, xstr] = sys ([], [], [], 0)
```

[x, u, y, dx] = trim ('sys', x0, u0, y0)为状态 x、输入 u 和输出 y 定义了各自的猜测值。在这种情况下，trim 命令就使[x - x0; u - u0; y - y0]的最大绝对值达到最小。

x, u, y 的每一个元素可以用下面的调用格式进行固定：

```
> >[x, u, y, dx] = trim ('sys', x0, u0, y0, ix, iu, iy)
```

整数向量 ix、iu 和 iy 指出对 x0、u0 和 y0 的哪一个元素进行固定。既然无法保证平

衡点一定存在,因此问题就转化为寻找一个稳态值使得

abs([x, (ix) - x0(ix) ; u(iu) - u0(iu) ; y(iy) - y0(iy)])

的最大绝对值达到最小。

trim 在限制状态倒数为零的情况下,用一个有约束的优化算法来求解一个由 x, u 和 y 的希望值所组成的最大绝对值最小问题。对于这样的一个问题,有可能没有可行解。在这种情况下,trim 就在状态倒数偏离 0 这种最坏条件下,使上面的最大绝对值达到最小。

要使状态微分固定为一个非零值,可以采用

```
>>[x, u, y, dx] = trim ('sys', x0, u0, y0, ix, iu, iy, dx0, idx)
```

其中 dx0 表示希望的偏离值,idx 用来表示对 dx 中的哪一个元素进行固定。

3. 举例

考虑如下一个线性空间模型

$$y = Cx + Du$$

假定这个模型的名字为"vtrim _ example",读者可以先用 Simulink 建立这个模型,如图 7-26 所示,模型中主要的模块就是 state space 模块,它在 continuous 库里。

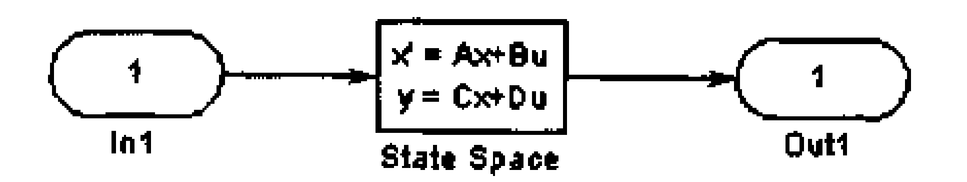


图 7-26 trim _ example 模型图

其中,模块的各个参数分别为:

```
A = [-0.09   -0.01   1   0]
B = [ 0      -7      0  -2]
C = [ 0       2      1  -5]
D = [-3       0      1   0]
```

例 1 要寻找一个系统的平衡点,可以用下面的命令:

```
>>[x, u, y, dx, options] = trim ('trim _ example')
x =
    0
    0
u =
    0
    0
y =
    0
    0
dx =
    0
```

```
    0
```

还可以求出所进行迭代的次数：

```
>> options (10)
ans =
    7
```

例 2　要寻找系统在 x = [1;1]和 u = [1;1]附近的一个平衡点，可以用下面的命令：

```
>> x0 = [1;1] ;
>> u0 = [1;1]
>> [x, u, y, dx, options] = trim ('trim_example', x0, u0) ;
>> x, u, y, dx, options (10)
x =
    1.0e-013
    -0.2770
    -0.2771
u =
     0.3333
    -0.0000
y =
    -1.0000
     0.3333
dx =
    1.0e-013
     0.9783
    -0.0053
ans =
    31
```

例 3　要寻找一个系统输出固定为 1 的平衡点，可以用下面的命令：

```
>> y = [1;1] ;
>> iy = [1;2] ;
>>[x, u, y, dx] = trim ('trim_example', [], [], y, [], [], iy) ;
>> x, u, y, dx
x =
     0.0009
    -0.3075
u =
    -0.5383
     0.0004
y =
```

```
    1.0000
    1.0000
dx =
    1.0e-016*
    0.0043
    0.4369
```

要寻找一个系统输出固定为1、状态倒数分别为0和1大的平衡点,可用下面的命令:

```
>> iy = [1;2] ;
>> dx = [0;1] ;
>> idx = [1;2] ;
>>[x, u, y, options] = tirm ('trim_example', [], y, [], [], iy, dx, idx) ;
>> x, u, y, dx
x =
     0.9752
    -0.0827
u =
    -0.3884
    -0.0124
y =
    1.0000
    1.0000
dx =
    -0.0000
     1.0000
>>options (10)
ans =
    13
```

第8章　深入理解Simulink

8.1　Simulink如何工作

经过前面的学习，大多数的仿真问题，读者都可以应付了，但是要想灵活、高效地使用Simulink，就必须对它的工作原理有些了解。尽管Simulink的初衷是为用户屏蔽掉许多繁琐的编程工作，而把主要精力放在模型的构建上。

8.1.1　基本模型

简单地说，Simulink里的每一个模块都是第1章中所说的一个系统，它有输入、输出和状态三个基本元素。在Simulink里，模块都是用向量来表示这三个元素的，本书分别用**u**、**x**和**y**来标记输入、状态和输出向量。图8-1反映了这个关系。



图8-1　Simulink模块的基本模型

在图中的三个基本元素中，状态向量无疑是最重要的，也是最灵活的一个概念。何谓状态呢？读者首先想到的可能是在线性系统理论里用来表示连续方程用的状态空间方法里的状态，在那里，状态被定义成一些微分。可以说那里的状态只是状态这个概念内涵的一部分，不仅仅只是工程系统存在状态，事实上，状态在非工程系统也能找到它的对应物。例如，前面讲到的排队系统中，队列中的等候人数等都可以被理解为状态。

确切地说，状态决定了模块的输出，而它的当前值是前一个时间的模块状态和(或)输入的函数。拥有状态的模块必须保存前面的状态值，并计算出当前的状态值。具有状态的模块也就是拥有保存以前状态值和后者输入值的存储空间。Simulink的Integrator模块是有状态模块的一个例子，Integrator模块输出是输入信号从仿真开始时到当前时刻的积分值。当前积分值依赖于Integrator模块的输入的历史记录，因此积分值是模块的一个状态。而Gain模块则是无状态模块的例子。Gain模块的输出完全由当前的输入值和增益决定，因此，Gain模块没有状态。

在Simulink里状态向量可以分为连续状态、离散状态以及两者的结合。输入、输出和状态这三个量的关系可以用下面的方程来反映。

$$\boldsymbol{y} = f_0(t, \boldsymbol{u}, \boldsymbol{x})$$

$$\boldsymbol{x}_{d_{k+1}} = f_u(t, \boldsymbol{u}, \boldsymbol{x})$$

$$\boldsymbol{x}_c = f_d(t, \boldsymbol{u}, \boldsymbol{x})$$

其中，
$$x = \begin{bmatrix} x_c \\ x_{dk} \end{bmatrix}$$

上面三个式子的意义是十分直观的，无论是对于连续系统还是对于离散系统，在用计算机进行仿真时，要估算一个系统的输入、输出以及状态向量，都是在采样时间点进行的，也就是仿真时间步。在每一个采样时刻，Simulink根据当前的时间、输入和状态来决定该采样时刻的输出，这个关系用fo来表示。对于离散状态，则要根据以前的状态来计算当前状态，对于一个行为复杂的系统，离散状态的更新完全有可能和连续状态有关系，所以式中是指整个系统的状态。而对于连续状态则是计算出连续状态当前的微分值。也就是说，任何一个模块的行为，都可以由上式的三个函数来刻画，设计一个模块，也就是要确定这三个函数，当然首先要确定三个向量输入、输出和状态的意义。

8.1.2 进行仿真

Simulink仿真包括两个阶段：初始化和模型执行。

1. 初始化

在初始化阶段，要完成的工作有：

(1)模块参数传给MATLAB进行估值，得到的数值结果将作为模块的实际参数。

(2)模型的各个层次被展开。每一个不是条件执行的子系统被它所包含的模块替代。

(3)模型中的模块按更新的次序进行排序。排序算法产生一个列表确保具有代数环的模块在产生它的驱动输入的模块被更新后再更新。当然，这一步也就是要先检测出模型中存在的代数环。

(4)决定模型中没有显式设定的信号属性，例如名称、数据类型、数值类型以及大小等，并且检查每个模块是否能够接受连接到它们输入端的信号。

Simulink使用一个名为属性传递的过程来决定未被设定的属性，这个过程将源信号的属性传递到它所驱动模块的输入信号。

(5)决定模型中所有没有显式设定采样时间的模块的采样时间。

(6)分配和初始化用于存储每个模块的状态和输出的当前值的存储空间。

2. 模型执行

完成这些工作之后就可以运行仿真了。一个模型是使用数值积分来仿真的。所运用的仿真解法器(仿真思想)依赖子模型提供它的连续状态的微分能力。计算微分可以分成两步来进行：首先，按照排序所确定的次序计算每个模块的输出；然后再根据当前时刻它的输入、状态来决定状态的微分，得到微分向量后再把它返回给解法器，后者用它来计算下一个采样点的状态向量。一旦新的状态向量计算完毕，被采样的数据源模块(sine wave模块等)和接收模块(scope模块等)才被更新。

在仿真开始时，模型设定待仿真系统的初始状态和输出，在每一个时间步，Simulink计算系统的输入、状态和输出，并更新模型来反映计算出的值。在仿真结束时，模型得出系统的输入、状态和输出。

在每个时间步，Simulink所采取的动作依次为：

(1)按排列好的次序，更新模型中模块的输出。Simulink通过调用模块的输出函数计算模块的输出。Simulink把当前值、模块的输入和状态传给这些函数计算模块的输出。

对于离散系统，Simulink 只有在当前时间是模块的采样实际的整数倍时，才会更新模块的输出。

(2)按排列好的次序，更新模型中模块的状态。Simulink 计算一个模块的离散状态的方法是，调用模块的离散状态更新函数。而对于连续状态，则是对连续状态的微分(在模块可调用的函数里，有一个用于计算连续状态的微分)进行数值积分来获得当前的连续状态。

(3)额外地检查模块连续状态的不连续点。Simulink 使用一种称为过零点检测(zero crossing detection)来检测连续状态的不连续点。

(4)计算下一个仿真时间步的时间。这是通过调用模块获得下一个采样时间函数来完成的。

3. 决定模块更新次序

在仿真中，Simulink 更新状态和输出，都要根据事先确定的更新次序，而更新次序对仿真结果的有效性非常关键。特别是，当模块的输出是它当前时刻的输入值的函数，那么这个模块必须在驱动它的模块被更新之后才能被更新。否则，模块的输出将是没有意义的。

这里请读者不要把被保存到模型文件的次序和仿真过程模块被更新的次序相混淆。Simulink 在模型初始化将模块排好正确的次序。

为了建立有效的更新次序，Simulink 根据输出和输入的关系，将模块分类。其中，当前输出依赖于当前时刻输入的模块称为直接馈入，所有其他的模块都称为非虚拟模块(关于直接馈入，见 8.1.4 节)。直接馈入模块的例子有 Gain，Product 和 Sum 模块，非直接馈入模块的例子有 Integrator 模块(它的输出只依赖于它的状态)，Constant 模块(没有输入)，Memory 模块(它的输出只依赖于前一个时间步的输入)。

基于上述的分类，Simulink 使用下面的两个基本规则对模块进行排序：

(1)每个模块必须在它驱动的模块中的任何一个之前被更新。这条规则确保模块在被更新时，它的输入有效。

(2) 非直接馈入模块可以按任何的次序更新，只要它们在它们所更新的直接馈入模块之前更新。这条规则可以通过把所有的非直接馈入模块以任何次序放在更新列表来满足。因此，它允许 Simulink 在排序过程中忽略非虚拟模块。

在排序过程中，Simulink 检查和标记代数环的出现。所谓代数环是指直接馈入模块的输出直接或者通过别的直接馈入模块连到模块的一个输入的信号环路。这样一个环路在更新模块时会产生一个死锁。然而代数环可以用于表达一系列的代数方程，这里把模块的输入和输出作为未知数。并且，这些方程在每个时间都要有解。因此，Simulink 假定包含直接馈入模块代表一些有解的代数方程，并在每次模块被更新时，解出这些方程。

另外一个约束模块更新次序的因素是用户给模块设定优先级，Simulink 在低优先级模块之前更新高优先级的模块。

8.1.3　过零检测

Simulink 用过零检测来检测连续信号的不连续的地方。过零检测在以下各个方面扮演着重要的角色。

(1)状态事件的获取;

(2)不连续信号的精确积分。

1. 状态事件的获取

一个系统发生一个状态事件,是指系统的某个状态值发生了能使系统产生显著变化的变化。状态事件的一个简单例子就是和地板相撞来回反弹的球。要对这样一个系统进行仿真,解法器不可能精确地使仿真步与球和地板接触的时间重合。因此,球就像是穿过了接触点,或者说球穿透了地板。

Simulink 使用过零点检测使仿真步精确地(在机器精度范围内)发生于状态事件发生的时刻。于是因为仿真的时间步准确地取在接触的时刻,所以仿真就不会产生穿透现象,并且速度从负到正的转换非常迅速。Simulink 里有一个弹球的演示模型,如图 8-2 所示。读者可以在 MATLAB 命令窗口输入 bounce 来演示它,或者也可以在 MATLAB 的 demo 窗口直接寻找。

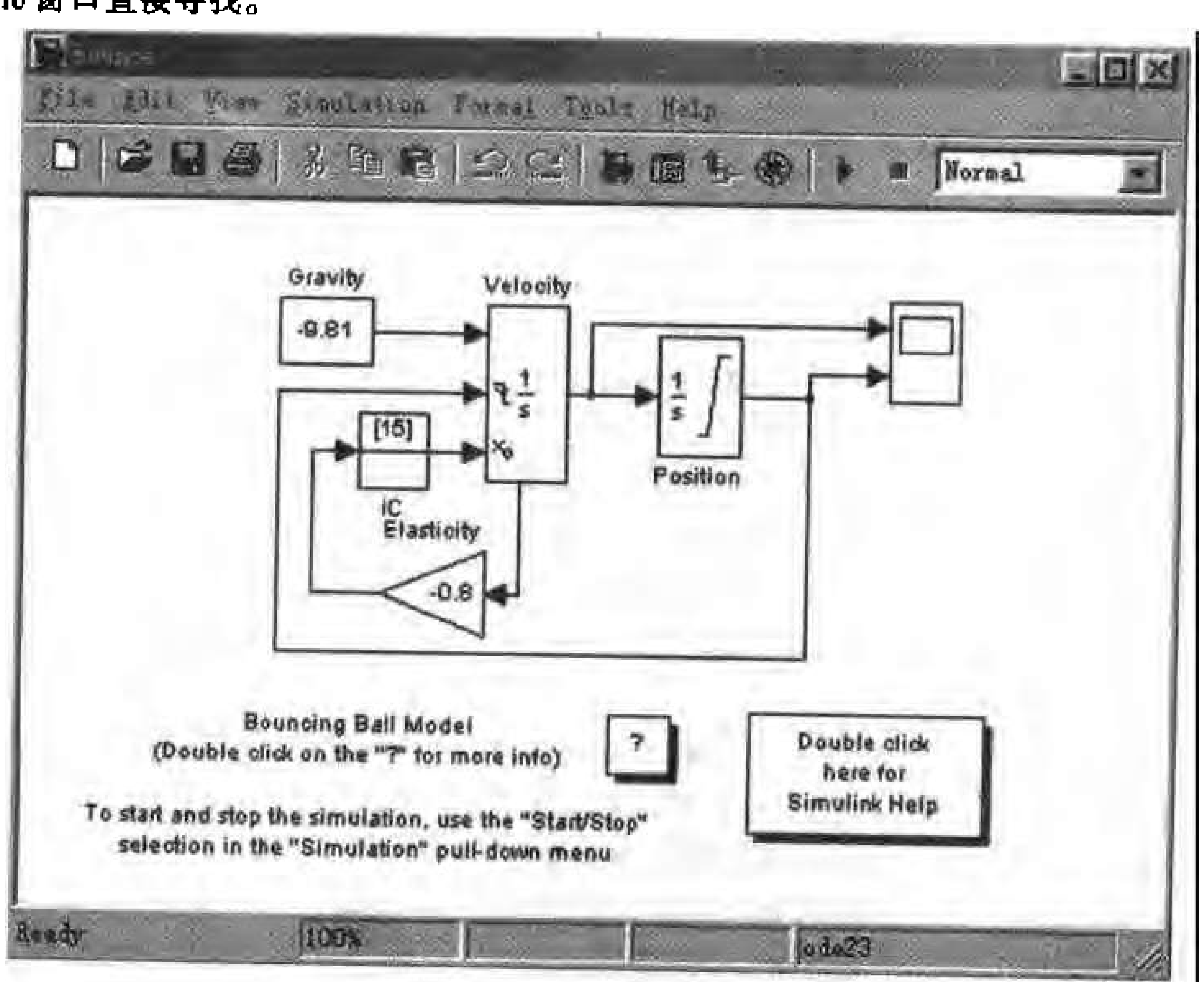


图 8-2 bounce 模型图

运行仿真,可以看到如图 8-3 所示的结果,上面一幅是小球速率的变化曲线图,下面一幅是小球运动过程的曲线图。

下面对这个演示模块进行简单的说明。它模拟以一定的初速度向上弹的小球,反复与地板相撞并最终变成零的过程。仿真模型有两个基本的假设:

第一个假设,不考虑空气阻力。在这个假设下,球在空中运动过程中的机械能量是不变的,也就是说球在与地板撞击后的瞬时速率和下次与地板撞击前的速率是相同的。

第二个假设,就是对球和地板的撞击过程进行了简化。在这里,模型假定撞击后的速

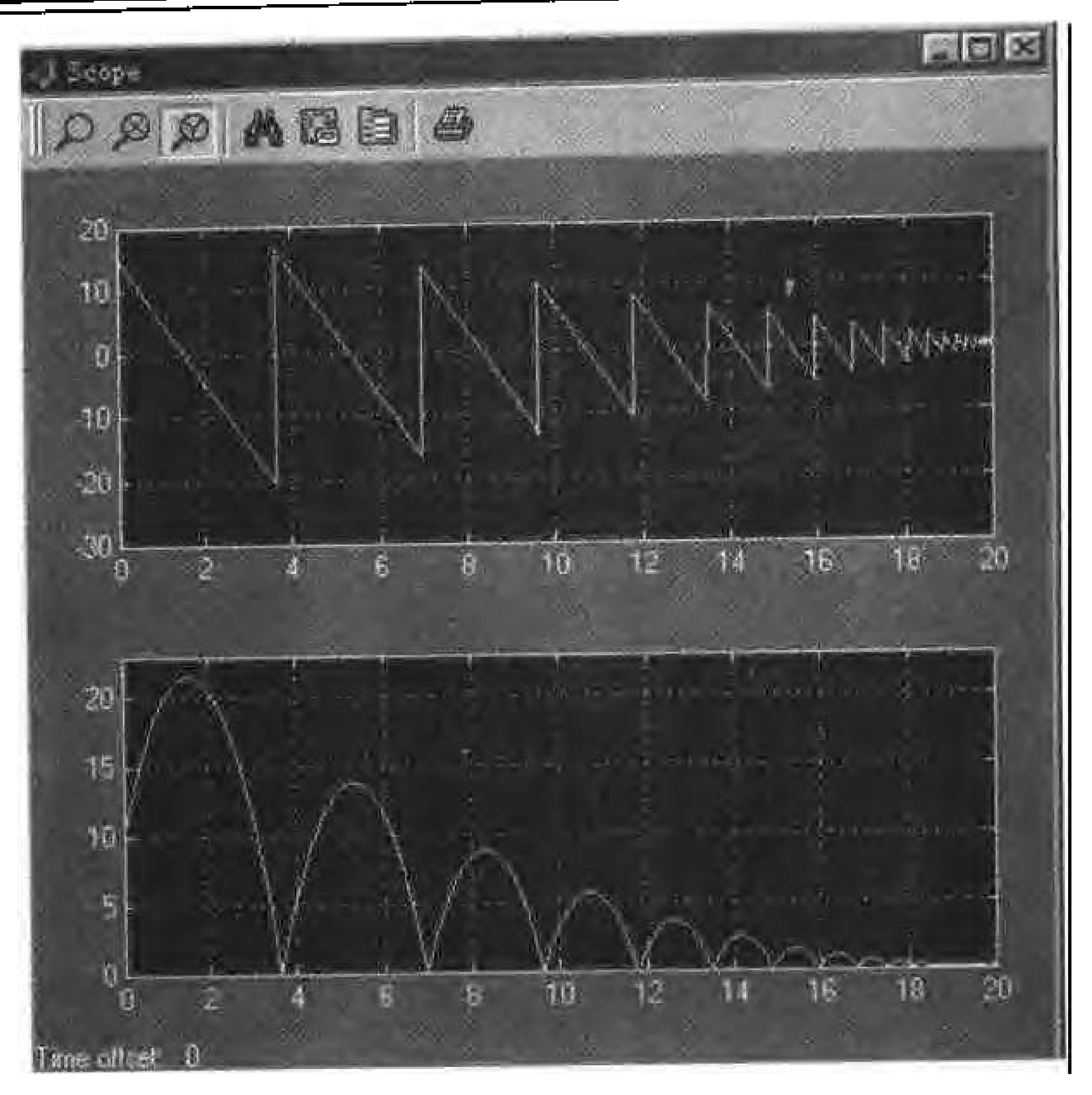


图 8-3 仿真运行结果

度与撞击前的速度之比始终是一个常数，在上图中是 -0.8。

明白了这两个假设，下面来看看用 Simulink 建立该模型的具体技巧。根据牛顿运动定理，不难理解速度和加速度之间的积分关系以及位移和速度间的积分关系。所以在模型的实现里有两个积分器——名称分别是 Position 和 Velotcity。对于 Position 模块，它的输入正是速度信号，并且由于球的初始位置是 0，所以它的初始状态是 0。此外，球与地板接触与否对应唯一的运算规律没有任何影响，所以 Position 积分器的积分限是 0 到无穷大。但是对于 Velocity 积分模块，由于小球和地板接触之后速度改向，并且使积分初始值变化，所以结构和常用的积分模块不太一样。从图中可以看出，这个模块有三个输入和两个输出。

请读者双击这些模块来看看它的参数对话框，就不难发现这些多余的输入和输出端口是如何添加的。图 8-4 是它的参数对话框。

External reset 参数，表示由外部信号使积分重新回到初始值，选择它的效果就是 Position 模块中类似于下降触发器的标志。而 Initial condition source 参数在这里被设为了 external，这表示积分的初始值可以由外部输入，一个好处就是可以动态地改变积分的初始值，选择它的效果就是 Position 模块左下角增加了个输入端口。这个端口由一个 Initial condition 模块来驱动。而检查框 Show state port 被选中的直接效果就是右下角的输出端，它输出积分的状态，对积分器而言就是积分的输出，这个输出经过一个增益为 -0.8 的 Gain 模块输入到 Initial condition 模块。实际上 Gain 模块就模拟了小球与地板接触后对速度的影响。

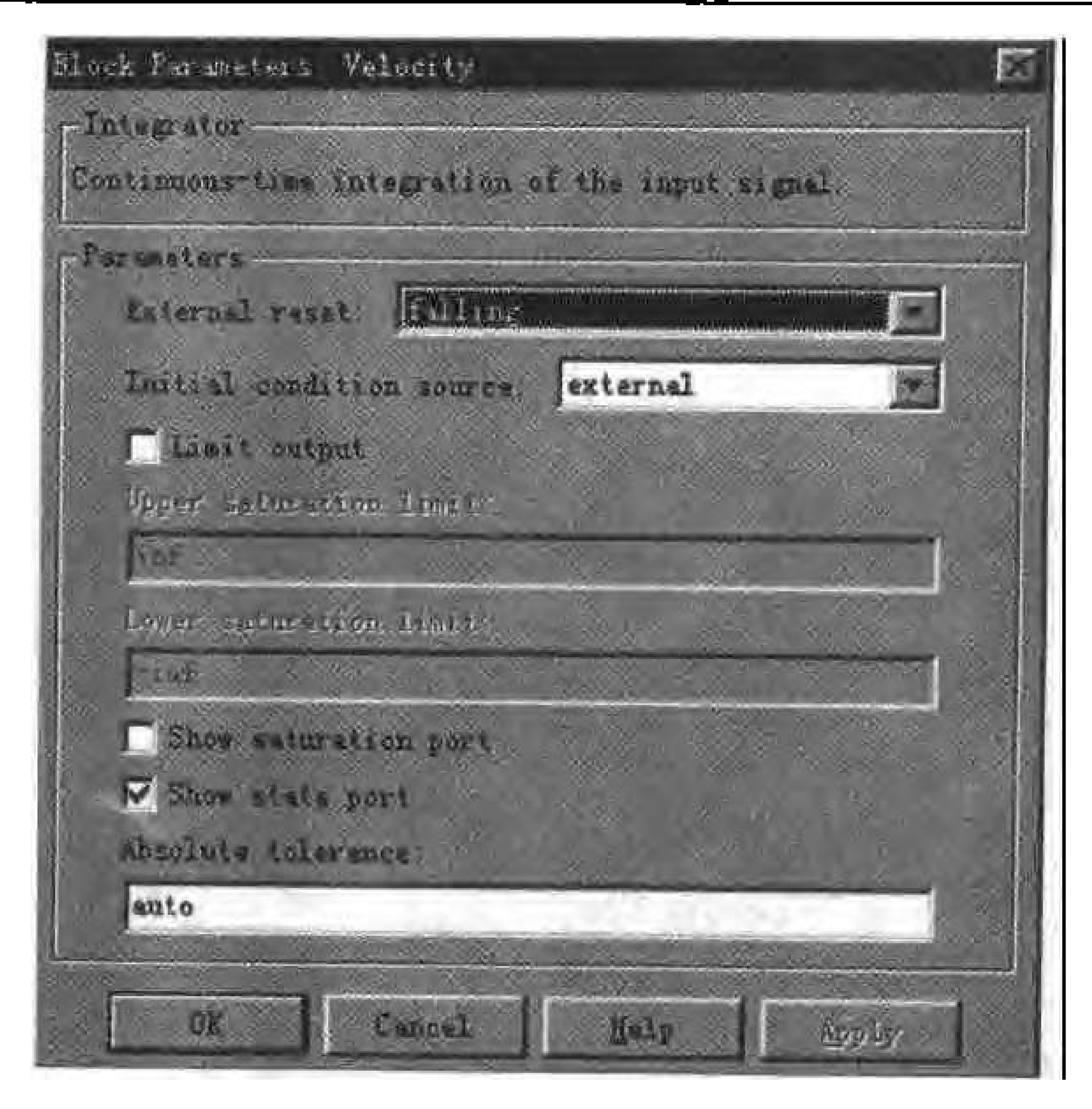


图 8-4 参数对话框

这个模型的最大的困难就是如何确定小球与地板的接触，在 Simulink 里是通过过零检测来解决它的。读者可以看到，Position 积分器的输出输入到 Velocity 的外部 reset 端口，因为该端口是下降触发的，一旦位移从正变为负就产生了一个下降触发事件，Simulink 可以通过过零检测来捕获这个事件。于是一旦检测到下降触发，就会使 Velocity 重新置为初始值，而此时的初始值为当前状态值乘以 -0.8，这就和物理过程符合了。

请运行仿真，来看看仿真的结果。

2. 不连续信号的积分

数值积分方法是建立在所积分的信号是连续的，并且具有连续的微分的假设下。如果在一个积分过程中遇到了不连续点(状态事件)，Simulink 使用过零检测来寻找不连续点发生的时间。而这次积分的上限只取到不连续点的左边沿。最后，Simulink 略过了不连续点，并对信号的下一个分段连续进行积分。

Simulink 模块中使用过零检测的一个例子是 Saturation 模块。过零点检测 Saturation 模块中的这些事件：

(1)输入信号到达上限；

(2)输入信号离开上限；

(3)输入信号到达下限；

(4)输入信号离开下限。

定义了自己的状态事件的 Simulink 模块被认为是具有固有的过零点。如果用户需要一个过零点事件的显示通知，可以使用 Hit crossing 模块。

状态事件的检测依赖于一个内部过零点信号点的构建。这个信号是不能被模块图表理解的。就以 Saturation 而言，它用于检测上限的过零点信号是 zcSignal = UpperLimit - u，其中 u 是输入信号。

过零点信号有一个方向属性，它的取值有三种：

(1) rising——当一个信号上升到零或者穿过零，或者离开零并且变成正数时发生的过零点；

(2) falling——当一个信号下降到零或者穿过零，或者离开零并且变成负数时发生的过零点；

(3) either——rising 或者 falling 其中有一个发生时就发生。

对于 Saturation 模块的上限而言，过零点的方向是 either。这样使得信号进入饱和或者离开饱和的事件可以通过相同的过零信号来检测到。

误差限度的大小对过零点的检测有很大的影响。如果误差限度太大，Simulink 就有可能检测不到过零点。数学里有这样一条定律，对于连续信号，如果有两个点的符号相异，那么在这两点之间必然存在一个过零点。这个定律是 Simulink 检测过零点的数学基础，它通过检查一个仿真时间步的首尾两点的符号，来判断在该时间步是否存在过零点。但是如果误差限度太大，就导致仿真的时间步不是足够的小，有些过零点就不能检测出来了。图 8-5 显示了仿真时间步的大小对过零点检测的影响。

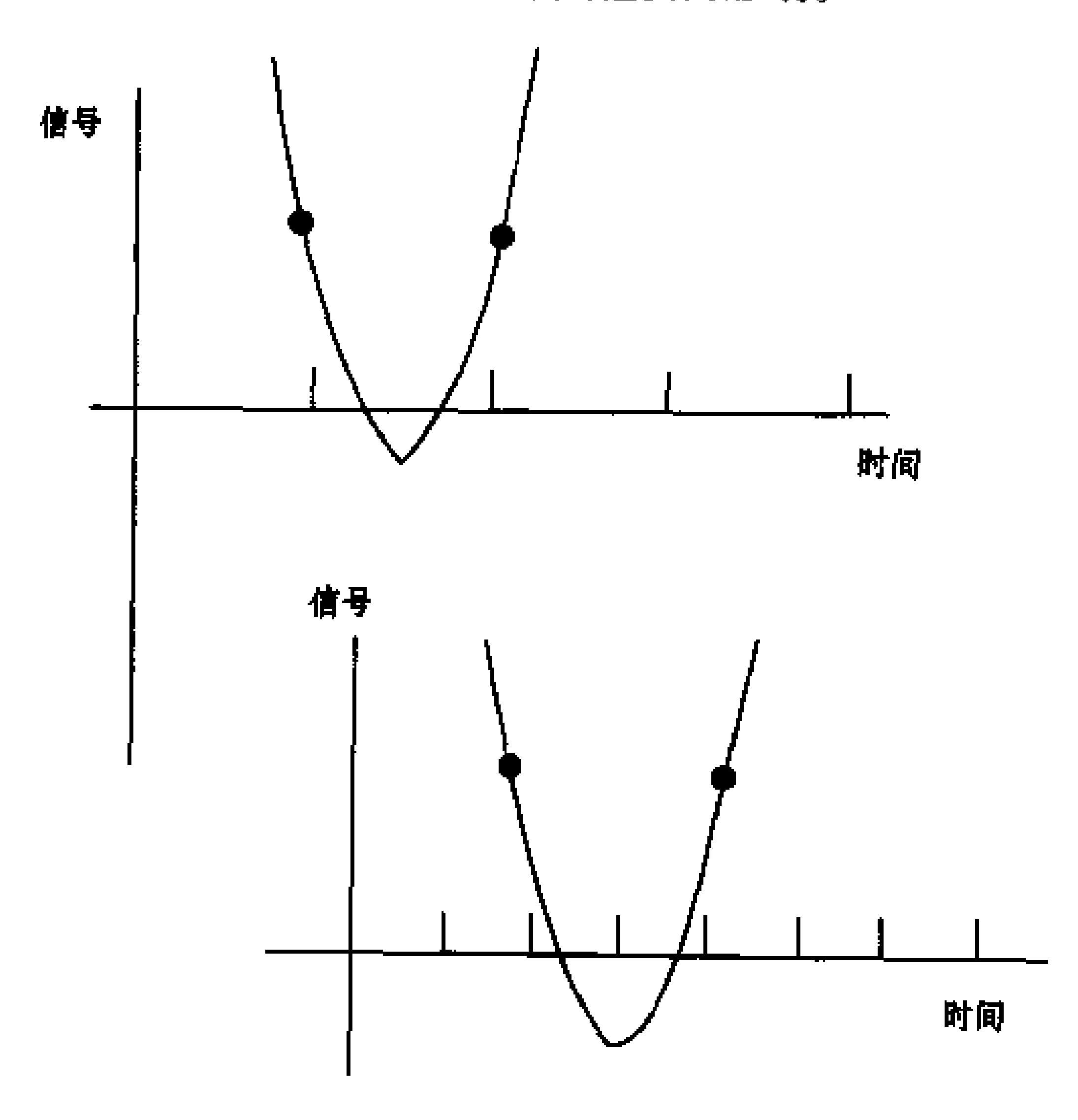


图 8-5 过零点检查示意图

在图 8-5 中，上图的仿真时间步取得较大，所以对于图中的信号在仿真步的首尾的采样值都是正数，没有符号变化，于是 Simulink 就会漏掉这两个过零点。而下图的仿真时间步缩小为原来的一半，就顺利地检查出这个过零点来。

为了防止这种情况的出现，Simulink 允许用户修改误差限度(见上一章)，通过缩小误差限度，可以使 Simulink 采用更小的仿真步长。

8.2 代 数 环

8.2.1 直接馈入环路(direct feedthrogh)——代数环

在 Simulink 里直接馈入的概念指是，具有直接馈入的模块在不知道输入端口的值的前提下无法计算输出端口的值。也就是当前时刻输出值的计算依赖于当前时刻的输入值，从模块的内部结构上说，模块内部存在延迟单元，在 Simulink 里，直接馈入的模块有：

(1)Elementary Math 模块；

(2)Gain block 模块；

(3)Integrator 模块的初始条件端口；

(4)Product 模块；

(5)有非零的 D 矩阵的 State - Space 模块；

(6)Sum 模块；

(7)分子和分母多项式具有相同阶数的 Transfer Fcn 模块；

(8)零点数和极点数相同的 Zero - Pole 模块。

前面几个模块为什么是具有直接馈入的模块，是很容易理解的。这里只略微提一下(4)，(6)，(7)三个模块。读者可以从库里找到 State - Space 模块，查看它的参数模块的描述信息。State - Space 模块的输出 y = Cx + Du，在 D 不为零时，输出 y 就和相同时刻的输入信号有关了，自然是直接馈入了。也就是说，直接馈入中的直接是指仿真时间意义上的，而不是指是否经过处理了，即和直接连线不是等同的。而对于 Transfer Fcn 模块和 Zero - Pole 模块附带的要求，都是为了保证模块所对应的传递函数具有常数项，这样当前时刻的输出就和当前时刻的输入有关了，所以也就有直接馈入。

对于大多数的模块，读者都可以按照上面的原则自行判断。

当具有直接馈入的端口由该模块的输出驱动时，或者是经过别的具有直接馈入的模块的反馈环路驱动，就发生了代数环。例如，图 8 - 6 所示的就是一个代数环的例子。

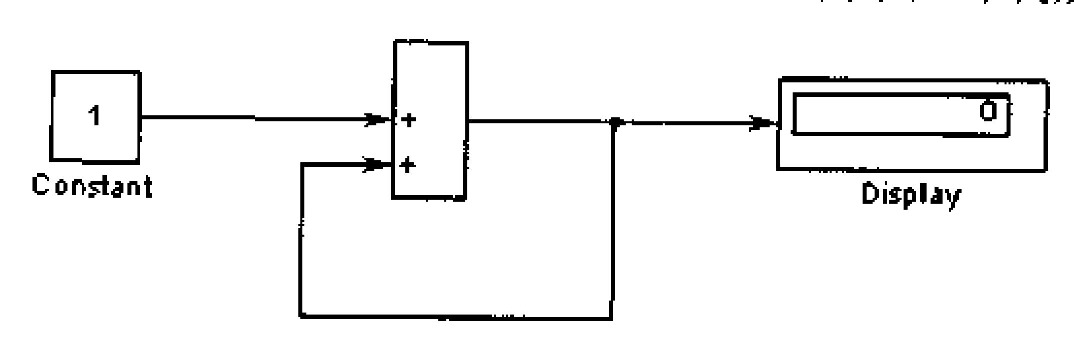


图 8 - 6 代数环示例

在图中，sum 模块是具有直接馈入的模块，它的输出直接连接到 sum 模块的输入端。于是整个模型所表示的意思就是一个方程了：

$$1 + u = u$$

当然，这个方程是无解的，所以当读者试图运行这个模块时，Simulinkk 就会显示错误信息。为此，可以在中间添加一个模块，就像图 8-7 所示的模型。

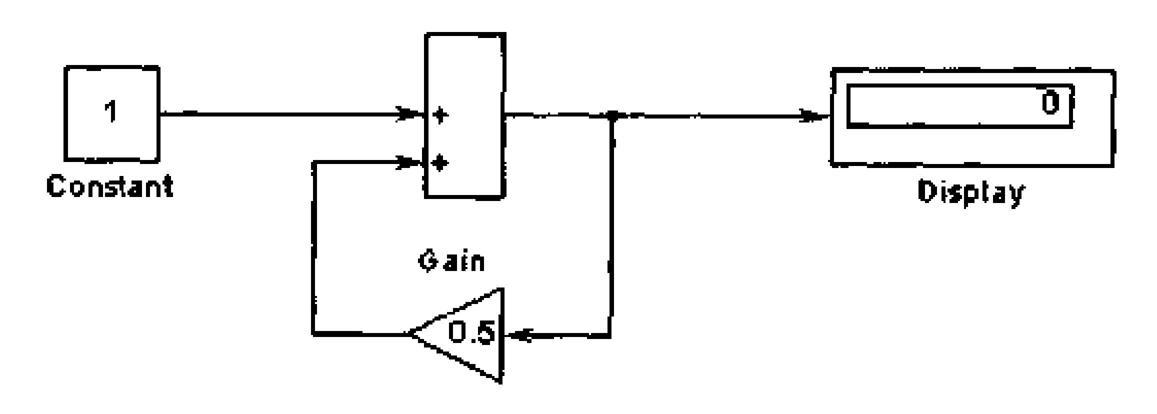


图 8-7　改进后的模型

在 Simulink，很容易建立具有多代数状态变量的向量代数环，下面的模型就是一个具有变量 z1 和 z2 的代数环，如图 8-8 所示。

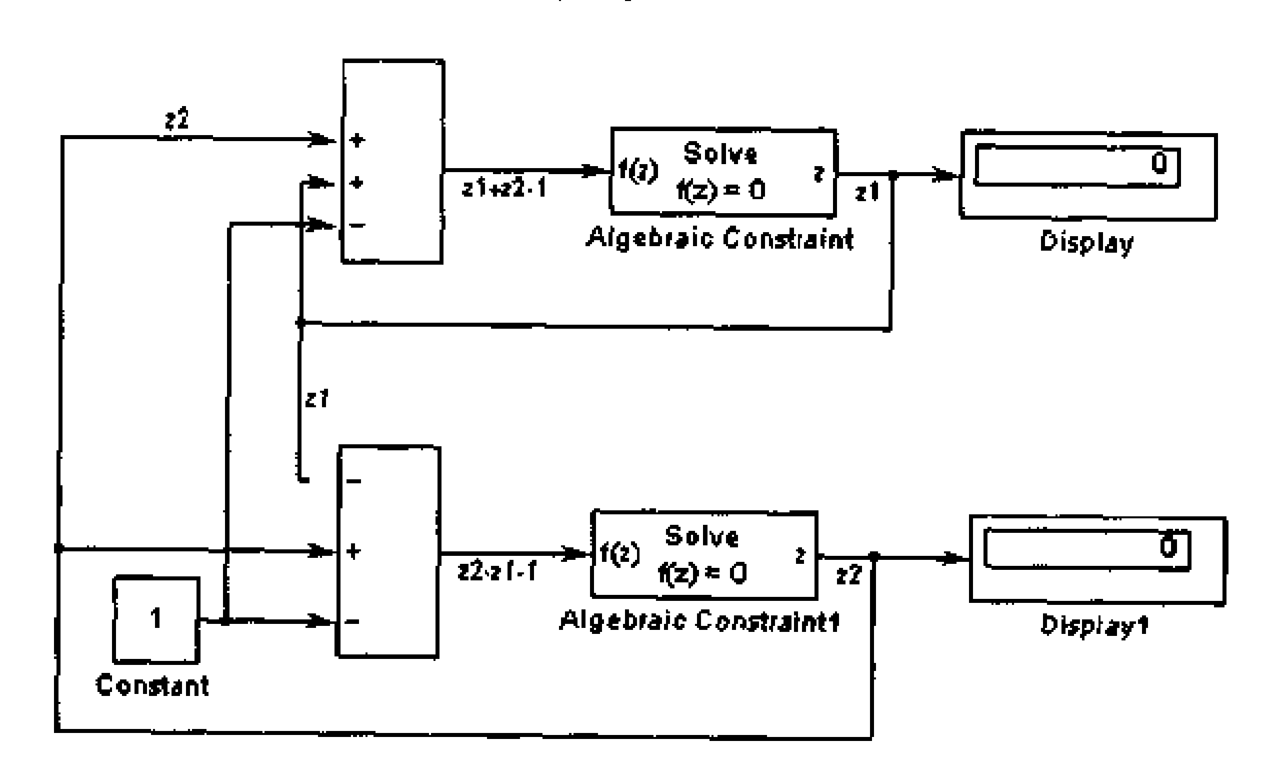


图 8-8　多状态代数环

这个模型的作用就通过建立一个向量代数环来解一个二元一次方程组：

$$\begin{cases} z_1 + z_2 - 1 = 0 \\ z_2 - z_1 - 1 = 0 \end{cases}$$

读者要注意模型中，Algebraic Constraint 模块的用法，它的位置是在 math 子库。Algebraic Constraint 是对代数方程建模的便捷方法，Algebraic Constraint 模块将它的输入信号 f(z)约束为零，并输出一个代数状态 z。注意，Algebraic Constraint 使用时，输出信号必须通过某种反馈途径影响输入信号，为了提高代数环解法器的效率，Algegraic Constraint 模块允许用户在参数对话框里设置它的初始猜测值。

当一个模型包含代数环时，Simulink 就在每一个时间步调用一个环路解法器的方法。环路解法器用迭代的方法来决定代数环所对应的方程。因此，具有代数环的模型比没有代数环的方程运行得慢。

为了解决 f(z)=0，Simulink 解法器使用具有弱的线性搜索的秩为 1 的牛顿方法更新偏微分雅可比矩阵。尽管这个方法是不很完善的，但是如果代数状态 z 没有一个好的初

始值估计,解法器就有可能不会产生一个收敛的值。除了可以通过参数对话框设置 Algebraic Constraint 代数状态初始值,但是这时,建立代数环解方程就必须使用 Algebraic Constraint 模块;也可以通过在连线上放置一个初始信号设置模块——IC 模块,来说明该连线的信号的初始值。

最后再强调一下,使用代数环时,应使用 IC 模块或者 Algebraic Constraint 模块来设置代数状态的初始猜测值。

8.2.2 非代数直接馈入环路

前面一节说过,一般情况下,包含直接馈入的环路都是代数环,这个通用法则也存在着例外:

(1)环路包含触发子系统。

(2)环路是一个模块的输出端口到积分器的重置端口。

在触发子系统的情况下,一个解法器假定了系统的输入信号在触发的时刻保持稳定。这就允许子系统使用前一个仿真时刻的输出结果来计算当前时间步的输入,这就意味着避免了使用代数环解法器。

8.3 离散时间系统

Simulink 具有对离散(采样数据)系统进行仿真的能力,模型可以是多速率的——模型包含的模块具有不同的采样速率,还可以是混合系统——既有离散模型,又有连续模块。

Simulink 里的每一个离散模块在输入端口都有一个内置的采样器,和一个零阶保持器在输出端。当离散模块和连续模块混合在一起时,在两次采样时刻之间,离散模块的输出保持一个常数。离散模块的输出只有在下一个采样时刻到来时,才会被更新。

在每一个离散系统的参数对话框中,都有一个 sample time 参数让用户设置模块的输出被更新的时刻。正常情况下,sample time 应该被设成标量,但是 Simulink 也允许用户通过给 sample time 参数设置一个两个元素的向量来说明一个偏移时间。具体的方法是在 sample time 编辑框里输入[ts, offset], ts 是指两次采样时间间隔,而 offset 则指偏移时间。离散模块被更新的时间满足下面的关系:

$$t = n * ts + offset$$

其中,n 是一个整数,而 offset 则是决定值比 ts 小的数,即正负均可。offset 在一个模块必须比别的模块早一些或者晚一些更新时非常有用。

注意,在仿真运行时,是无法改变一个离散模块的采样时间的,这和模块的其他参数不同,如果要改变 sample time,必须先停止仿真。

1. 纯离散系统

纯离散系统可以用任何一种解法器来进行仿真,它们所求到的结果没有什么区别。

2. 多速率系统

多速率系统是指系统里包含以不同速率采样的模块。它们既可以是纯离散系统,也可以是连续和离散的混合系统。图 8-9 就是一个多速率系统的例子。

模型用到了一个 Constant 模块、两个 Discrete Transfer Fcn 模块以及两个 out 端口模

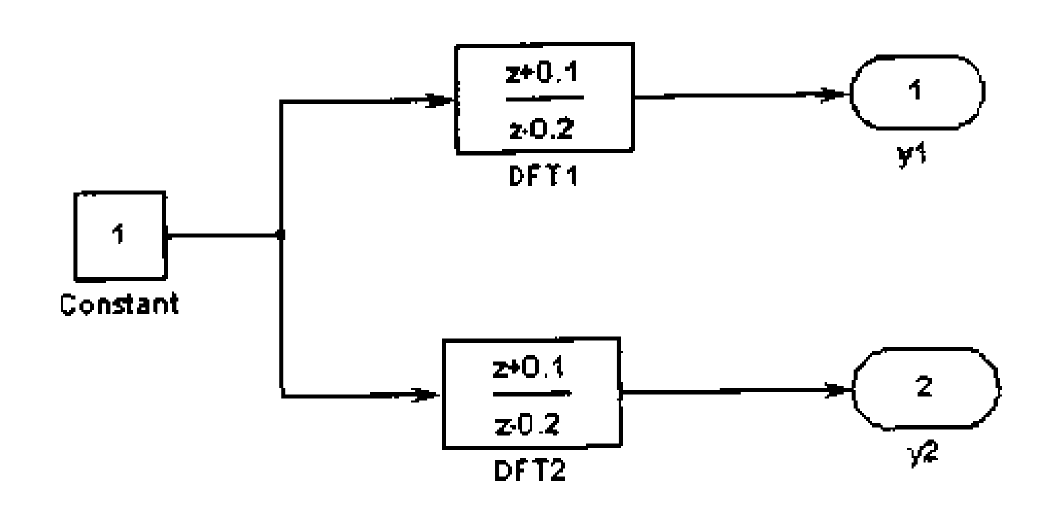


图 8-9 多速率系统示例

块,其中 Discrete Transfer Fcn 的位置在 math 子库。它们的传递函数都设置为:Numerator :[1 0.1],Denominator:[1 -0.2]。如图 8-10 所示。

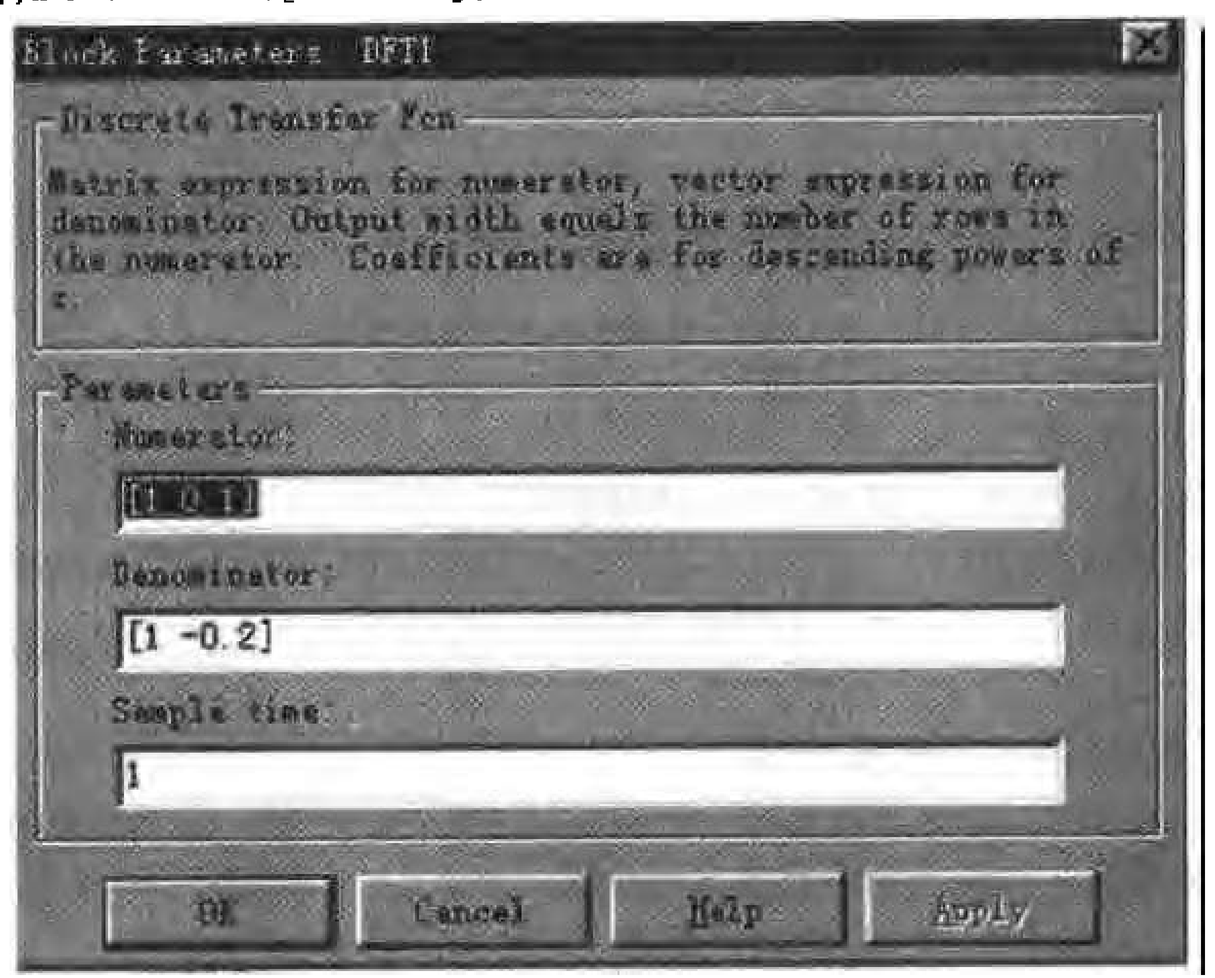


图 8-10 传递函数的设置

至于 Sample time 参数的设置,DFF 为[1 0.1],而 DFF2 为[1 -0.2]。

读者一定注意到了,模型中没有接收器类型的模块,而只有两个输出端口,这里用到了一项技术,Simulink 通过这两个端口把结果返回到 MATLAB 工作空间里的变量中。读者在 MATLAB 命令窗口输入下面的命令:

```
>>[t, x, y] = sim ('multirate', 5) ;
  %通过运行命令行 sim 执行仿真,其中 multirate 是要运行的系统的名称,而后面的参
  %数是指定仿真时间
>>stairs (t, y) ;
  %画出输出的阶梯图
```

画出的曲线,如图 8-11 所示。

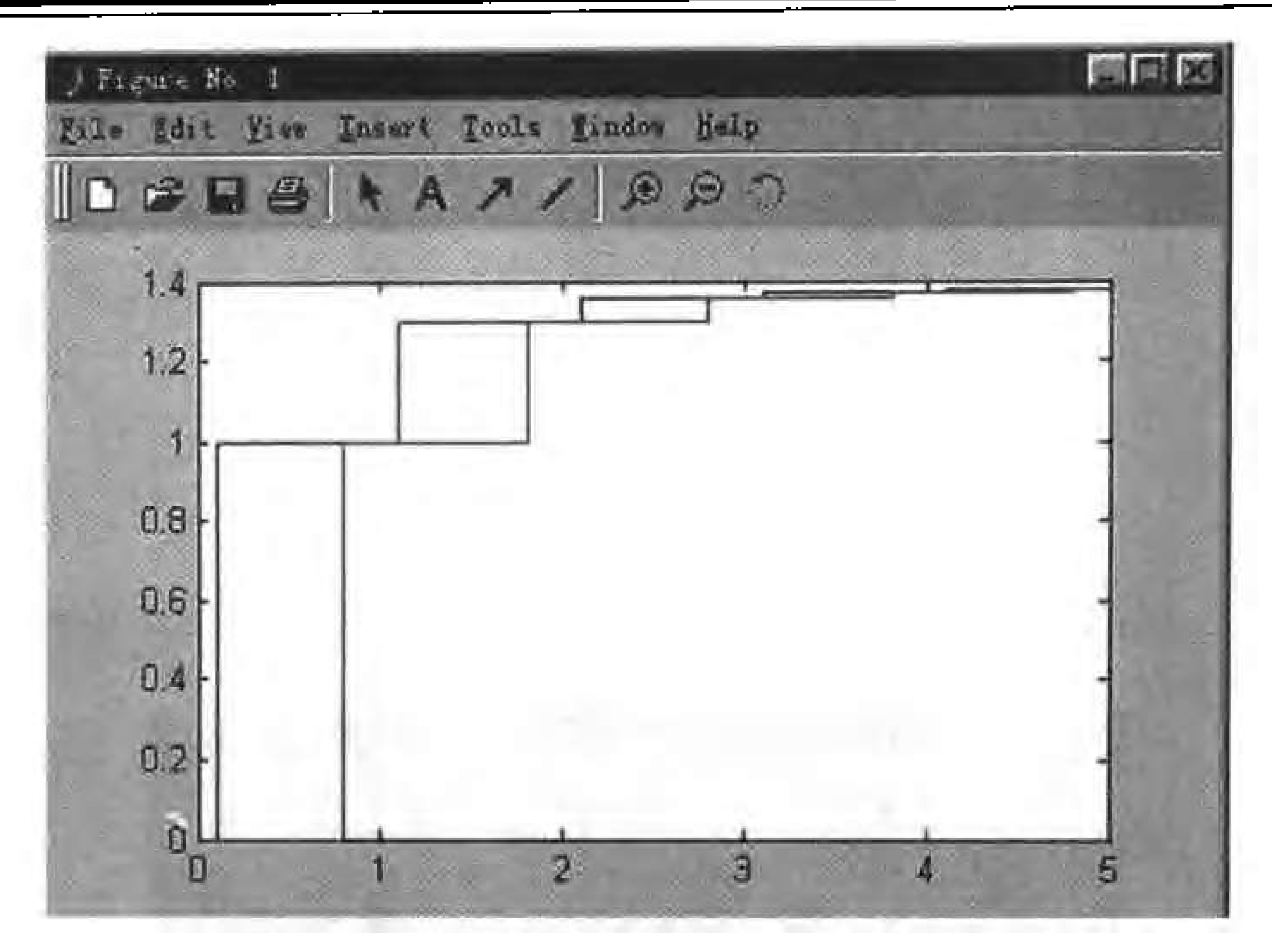


图 8-11 多速率系统的输出曲线

从图中可以看出：

(1)离散系统的输出都是柱状图形，也就是说在采样点之间，模块的输出保持不变。

(2)由于 DFF1 和 DFF2 的 Sample time 参数设置不同，所以它们发生跳变的时刻不同，在 y2 曲线上明显可以看出一个偏移，它的第一次跳变从大于 0 的时刻开始，而 y1 则从 0 时刻开始。

3. 用颜色来反映采用时间

在 Simulink 的 formate 菜单下有一个 sample rate colors 选项，它的作用就是用不同的颜色来反映模型中不同模块的采样速率的快慢。表 8-1 是不同的颜色所表示的意义。

表 8-1 采样实际的颜色表示

颜色	表示的意义
黑 色	连续模块
红紫色	常数模块
红 色	最快的离散采样时间
绿 色	次最快的离散采样时间
蓝 色	第三快的离散采样时间
亮 蓝	第四快的离散采样时间
暗 绿	第五快的离散采样时间
蓝 绿	受触发的离散采样时间
灰 色	固定的最小的采样时间
黄 色	混合模块(组合模块的子系统或者是组合信号的 demux 和 mux 模块，它们的组成元素有不同的采样速率)

读者注意，就像上表所示的一样，Simulink 为各个模块设置的颜色不是指绝对的采样时间，而是指向对于其他模块的采样时间的快慢。为了让读者更好地理解 Simulink 的这个特性，下面我们来介绍 Simulink 的采样时间传播引擎（STPE）。图 8－12 演示了一个采样时间为 Ts 的 Discrete Filter 模块驱动一个增益模块的情形。图 8－13 是它的运行结果。

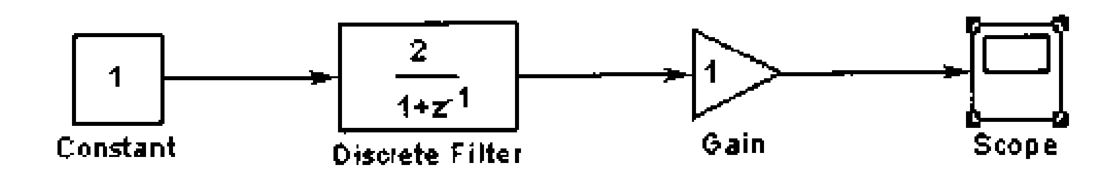


图 8－12 采样时间

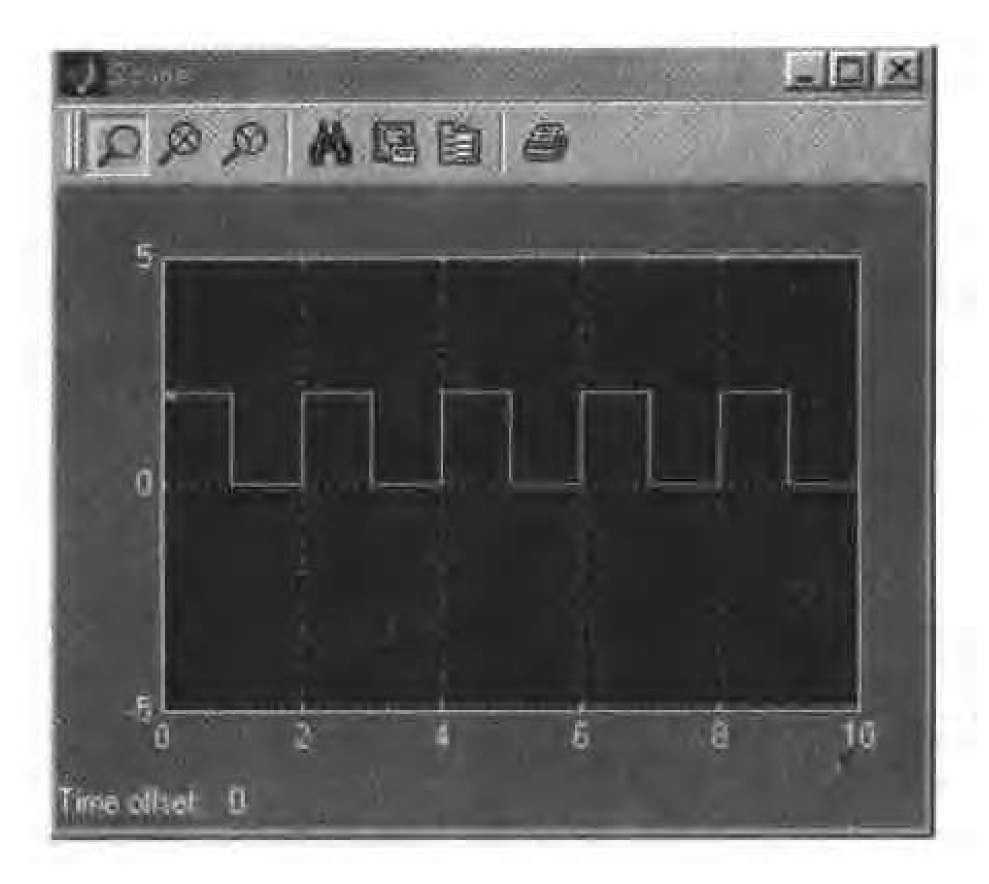


图 8－13 运行结果

因为增益模块的输出仅仅是简单地把输入乘以一个常数，所以它的输出和离散滤波模块的输出以相同的速率更新，也就是说增益模块的有效采样速率等于滤波器的采样速率。这就是隐藏在 STPE 后面的基本机制。

建好模型后，即可以用 format 菜单下的 sample time feature 特性来看看采样实际的颜色特性。上面的这个模型比较简单，读者可以用 Simulink 的演示模型来试验这个特性，会看得更清楚。在模型改变时，Simulink 并不会自动地去更新各个模块的采样时间颜色，而是需要用户用 Edit 菜单下的 update the diagram 命令去更新。

Simulink 遵循以下法则为单个的模块设置采样速率：

（1）连续模块（例如积分模块、微分模块、传递函数模块等等），按定义，采样时间为连续的。

（2）常数模块（例如，常数模块），按定义，采样时间是常数。

（3）离散模块（如零阶保持模块，单位时延模块，离散传递函数模块），它们的采样时间由用户在模块对话框的显式说明来决定。

（4）其他的模块，它们的采样时间取决于输入信号的采样时间。比方说，一个 Gain 模块当它由一个积分模块来驱动时，就是一个连续模块；而在由 Zero－OrderHold 模块驱动

时，却被当成和一个 Zero – OrdreHold 模块相同的采样时间离散模块。

（5）具有不同的采样时间的输入信号的模块，如果所有的采样时间都是最快采样时间的整数倍，那么整个明显的仿真时间都被设置成最快的采样时间。这一点 很容易理解，Simulink 就可以保证每个输入信号所需要的采样时间点都能被产生。而在模块内部，Simulink 检查当前的仿真时间步是否和模块定义的采样时间重合，如果重合就更新模块的输出。如果可变步长解法器被使用，模块的采样时间就是连续时间。如果定长解法器被使用，而且所有信号的采样时间的最大公因子——基准时间——能够被计算，那么这就是模块的采样时间。

但是读者需要注意到 demux 和 mux 模块是组合信号的虚拟模块——信号保留原来的定时信息通过该模块。因此，如果 demux 模块引出的直线是具有不同的采样时间的源驱动，那么这些直线就会有不同的颜色。而 mux 和 demux 模块本身被赋成黄色，表示它们是混合的，具有多个速率的信号。

类似地，对于包含不同采样速率模块的子系统，也用黄色来表示，只有在子系统内的每个模块都在相同的采样速率工作时，Simulink 才会根据这个速率来确定子系统的颜色。

在某些情形下，Simulink 也能向后传递采样时间到源模块，如果这样做不会影响仿真的结果。例如，在下面的模型中（如图 8 – 14 所示），Simulink 识别出信号发生器模块驱动离散积分模块，所以 Gain 模块和信号发生器模块的采样时间都被设成离散积分模块的采样时间。

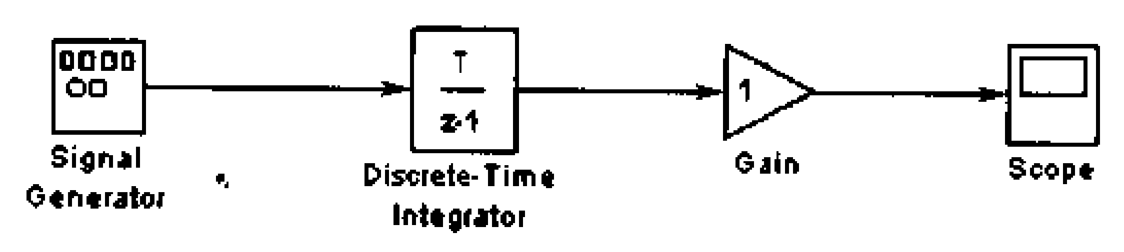


图 8 – 14　采样时间的传播

8.4　使用回调函数

8.4.1　回调函数基本概念

在具体讲解回调函数的用法和作用前，我们先来看看 MATLAB 中使用 callback 函数的一个演示程序。请读者在 MATLAB 命令窗口输入

> > f14 ;

f14 是 Simulink 提供的一个复杂的仿真模型示例，读者可以通过在 matlab demo 窗口手动找到这个模型。这里提及这个模型的目的不是为了介绍这个模型的原理、具体的构造技巧，主要是让读者体会一下什么是回调函数，它能起什么作用。图 8 – 15 是 f14 的模型图。

读者首先克服对 f14 模型的作用以及各个模块组成原理的好奇，把注意力放在本书下面要让读者思考的问题。在 f14 的许多模块的参数对话框里经常出现一些变量，而在模型中却找不到任何关于这些变量的定义。关于在参数对话框里使用变量的情形，本书在 Simulink 详解第 5 章“封装子系统”一节，在讲解参数传递的两种方式：evaluate 和 liter-

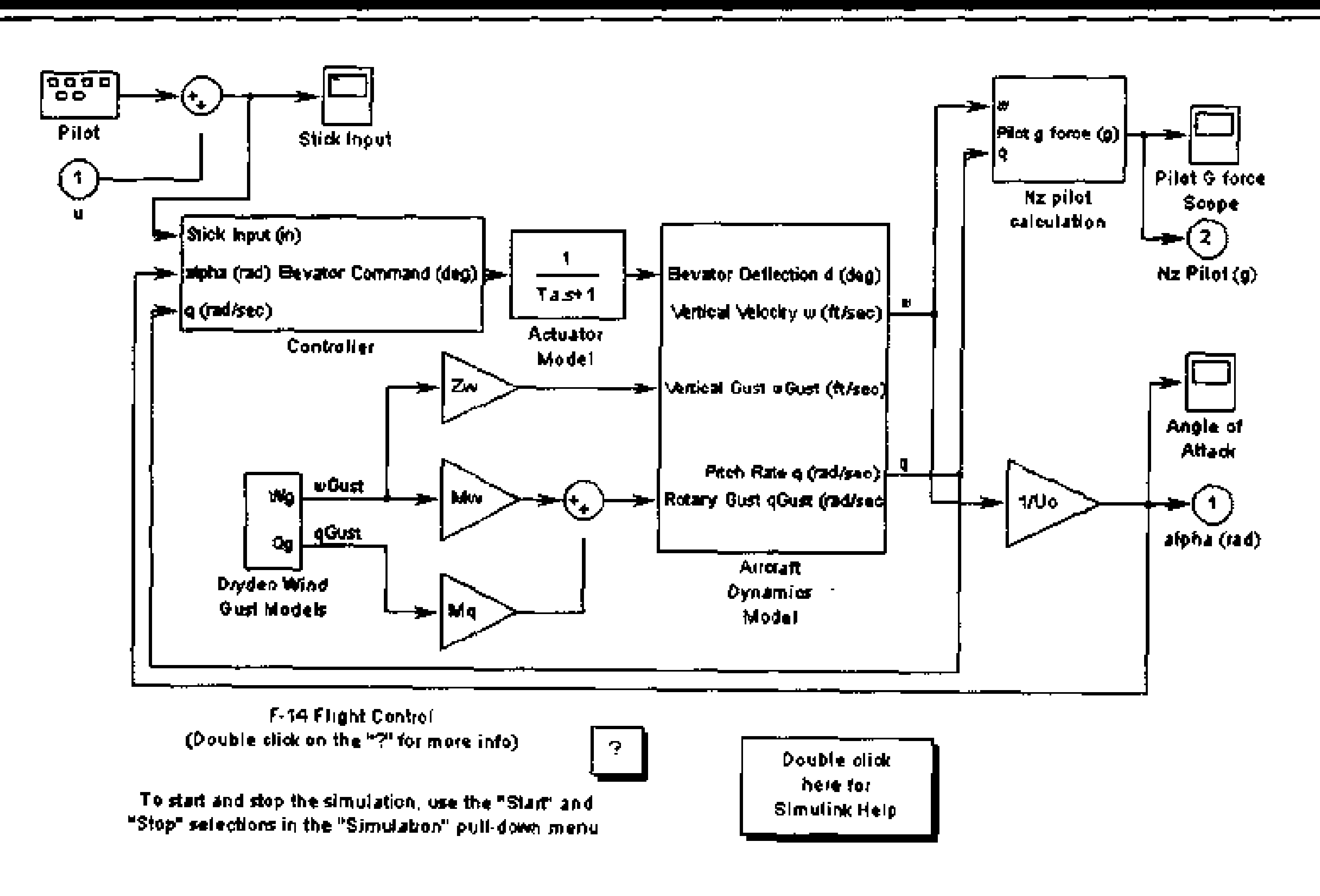


图 8－15　f14 模型图

al 时，就曾经举过例子。但是在那里，设置模块参数为变量时，在运行前都要在 MATLAB 工作空间先定义这个变量。因为按照 evaluate 模式（这是 Simulink 模块常用的参数传递模式）的原理，任何参数设置对话框输入的内容，首先被 MATLAB 进行估值，其实也就是相当于把它们输入在 MATLAB 命令窗口，求出这些表达式的值，再传递给模块。根据 MATLAB 的估值顺序，一个字符串首先被认为是 MATLAB 工作空间的一个变量，如果不是，就认为是内置函数，依此类推。所以在参数设置时，使用已经在 MATLAB 工作空间定义过的变量（其实可以使用一切可以估值的表达式）当然是可行的。但是在 f14 模型里，似乎不需要这样的过程。模型被打开后，无需在 MATLAB 里定义这些变量，至少表面上看来不需要，就可以顺利运行模型的仿真。图 8－16 是运行输出的结果。

真的是不需要事先定义这些变量吗？其实不然，试想如果没有定义，那 MATLAB 又从何估值呢？请读者在 MATLAB 命令窗口输入 who 命令查询变量。

```
>> who %查询变量
Your variables are :
Ka        Mq        Tal       W2        b
Kf        Mw        Ts        Wa        beta
Ki        Sa        Uo        Zd        cmdgain
Kq        Swg       Vto       Zw        g
Md        Ta        W1        a         gamma
```

也就是说，f14 中用到的一些变量其实都已经事先定义了，只是定义的方式不是以前的那种手工定义，而是通过自动调用执行某些 MATLAB 命令和函数来完成。读者可以以预计的方式在模型被打开时，调用某个程序或者函数。这种预期和客观的技术事实是基本吻合的，f14 这种功能的实现，依赖于 callback 函数（回调函数）。

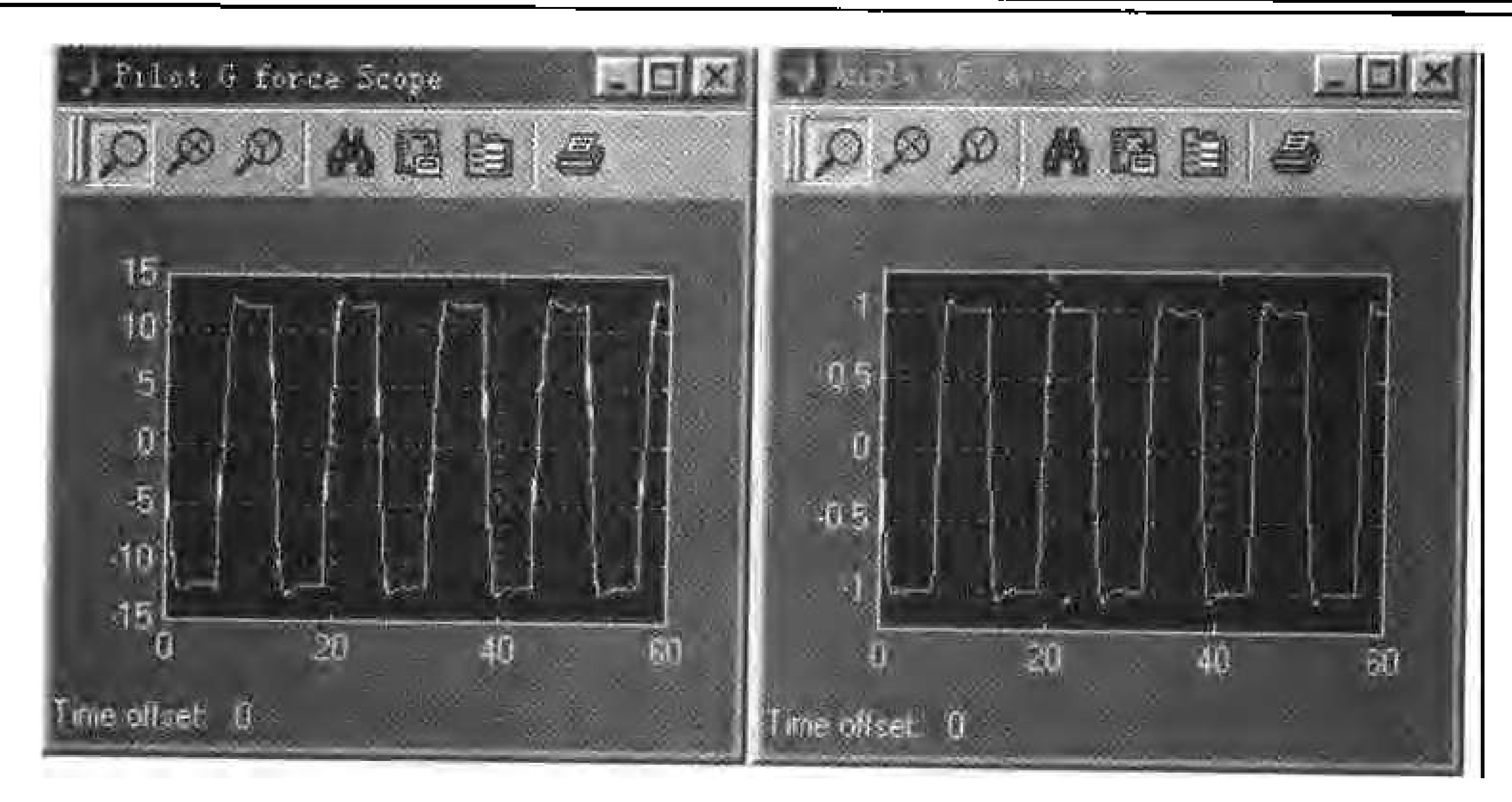


图 8-16 仿真运行输出结果

所谓回调函数,是指当用户定义的模型的图表或者模块发生某种特殊行为时,执行的 MATLAB 表达式(M 文件和 M 文件函数)。例如,在 f14 模型打开时,Simulink 就按照在模型建立时设置好的回调函数来执行定义变量的工作。回调函数有点像许多高级编程语言里的事件处理程序。在 MATLAB 里,为模型或者模块的某种行为设置回调函数的方法是,设置与该行为对应的参数为相应的回调函数名,这些参数就像高级程序语言里的事件。例如,要在模型被载入前,执行一段用于定义一些数据变量参数(如同 f14 模型)的 M 文件,就可以设置模型的 PreLoadFcn 参数为该 M 文件的文件名称。在 Simulink 里,为模型设置参数的方法是使用 set_param 命令。例如,要设置 PreLoadFcn 参数为'modeldat'这一 M 文件名,就可以使用下面的命令

```
>> set_param ('模型名称','PreLoadFcn', 'modeldat') ;
```

在举例说明如何使用 set_param 命令来设置回调方法参数之前,我们来检验一个 f14 模型的回调函数。F14 模型的 PreLoadFcn 回调方法参数真的被设置了一个回调函数吗? 在 Simulink 里可以用 get_param 命令来获得模型文件中某个参数的值。例如,这里可以在 MATLAB 命令窗口输入

```
>> get_param ('f14' , 'PreLoadFcn' )
ans =
f14dat
```

命令执行后,f14 模型的 PreLoadFcn 参数设置了一个名为 f14dat 的 M 文件(函数)。不难查找到这个 M 文件,它的位置是你的 MATLAB 安装目录下的 foolbox \ Simulink \ simdemos 目录。整个 M 文件的代码如下:

```
%Numerical data for F-14 demo
%   Copyright(c)1990-1998 by The MathWorks,Inc. All Right Reserved.
%   $ Revision :1.9 $
g = 32.2 ;
Uo = 689.4000 ;
Vto = 690.4000 ;
```

```
% Stability derivatives
Mw = -0.00592 ;
Mq = -0.6571 ;
Md = -6.8847 ;
Zd = -63.9979 ;
Zw = -0.6385 ;
% Gains
cmdgain = 3.490954472728077e-02 ;
Ka = -0.6770 ;
Kq = 0.8156 ;
Kf = -1.7460 ;
Ki = -3.8640 ;
% Other constants
a = 2.5348 ;
gamma = 0.0100 ;
b = 64.1300 ;
beta = 426.4352 ;
Sa = 0.005236 ;
Swg = 3 ;
Ta = 0.0500 ;
Tal = 0.3959 ;
Ts = 0.1000 ;
W1 = 2.9710 ;
W2 = 4.1440 ;
Wa = 10 ;
```

里面的命令定义了 f14 里用到的一些变量。于是，在 Simulink 载入这个模型之前，就会先调用 f14dat 这个 M 文件，在 MATLAB 工作空间里定义好用到的变量。这个参数设置的相应命令就是

```
>>set_param ('f14', 'PreLoadFcn', 'f14dat') ;
```

但是建议读者不要对 Simulink 的示例模型的参数随便进行试验，即使是修改，也请先对模型进行备份。比较好的方法是建立一个简单模型来试验它的用法。

至此，前面提到的 f14 模型中出现的疑问得到了比较完善的解释。

剩下的一个关键问题是，模型或者模块中有哪些参数用于定义回调方法，而这些方法又在什么时候调用。表 8-2 列出了这些回调参数以及对应的回调方法的执行时间。

表 8-2　模型的回调函数

回调函数名称	回调方法执行的时间
CloseFcn	在模块图表被关闭之前
PostLoadFcn	在模型被载入之后，为这个参数定义一个回调方法对产生一个要求模型已经存在的界面非常有用

（续）

回调函数名称	回调方法执行的时间
InitFcn	在模型的仿真开始之后
PostSaveFcn	在模型被保存之后
PreLoadFcn	在模型被载入之前，用于载入预先模型使用的变量
PreSaveFcn	在模型被保存之前
StartFcn	在模型仿真开始前
StopFcn	在模型仿真停止之前，在 StopFcn 执行前，仿真结果先被写入工作空间中变量和文件中
CloseFcn	当模块是使用 close_system 命令关闭时
CopyFcn	在模块被拷贝之后，这个回调对子系统是递归的（意思是，当用户复制一个子系统，它包含 CopyFcn 参数被定义的模块，那么回调方法也执行）。如果使用 add_block 命令复制模块，这个方法也被执行
DeleteFcn	在模块被删除之前，这个回调对子系统递归
DestroyFcn	当模块被毁坏时
InitFcn	在模块图表被编译和模块参数被估值之前
LoadFcn	当模块图表被载入，该回调对子系统递归
ModelCloseFcn	在模块图表被关闭之前，该回调对子系统递归
MoveFcn	当模块被移动或者调整大小时
NameChangeFcn	当模块的名称或者路径改变后，当子系统的路径被改变时，它递归的为子系统所包含的每个模块在调用它们本身的 NameChangeFcn 方法后，调用该方法
OpenFcn	当模块被打开时，这个参数一般用于系统模块。该方法在用户双击打开模块，或者使用以该模块作为参数的 open_system 命令时，被调用。OpenFcn 参数重载打开模块一般发生的行为，例如，显示对话框或者打开模块
ParantCloseFcn	在关闭包含该模块的子系统前，或者在该模块组成用 new_system 新建的子系统的一部分时
PreSaveFcn	在模块图表被保存前，该回调对于子系统模块是回调的
PostSaveFcn	在模块图表被保存后，该回调对于子系统模块是回调的
StartFcn	模块图表被编译之后，仿真开始之前
UndoDeleteFcn	当一个模块删除操作被取消时

下面就来举几个简单的示例模型来说明什么时候以及如何设置回调方法。

8.4.2 回调函数示例

这里将用两个例子来说明回调函数的使用。

1. 对变量进行初始化

这个例子基本上就是对前面 f14 模型的一个简单回顾，它使用一个回调函数'exam_542dat'来对模型 exam_542dat 进行变量初始化。而在模型结束时将仿真结果绘图。

请读者先按图8-17所示的图建立好模型，其中 Gain 模块的参数设置为 gain，请在模型仿真参数对话框的 Workspace I/O 页设置，把时间和输出结果保存到工作空间。如图8-18所示。

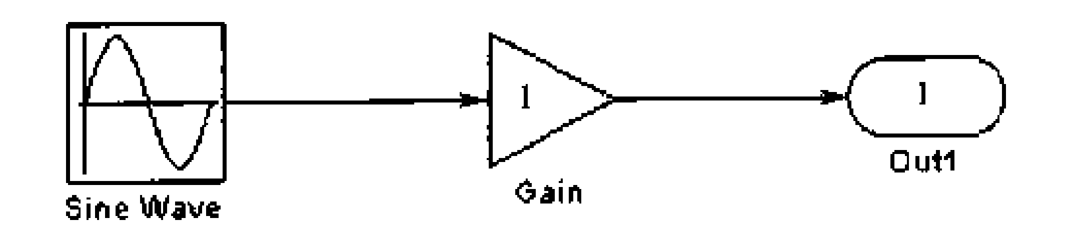


图 8-17 模型图表

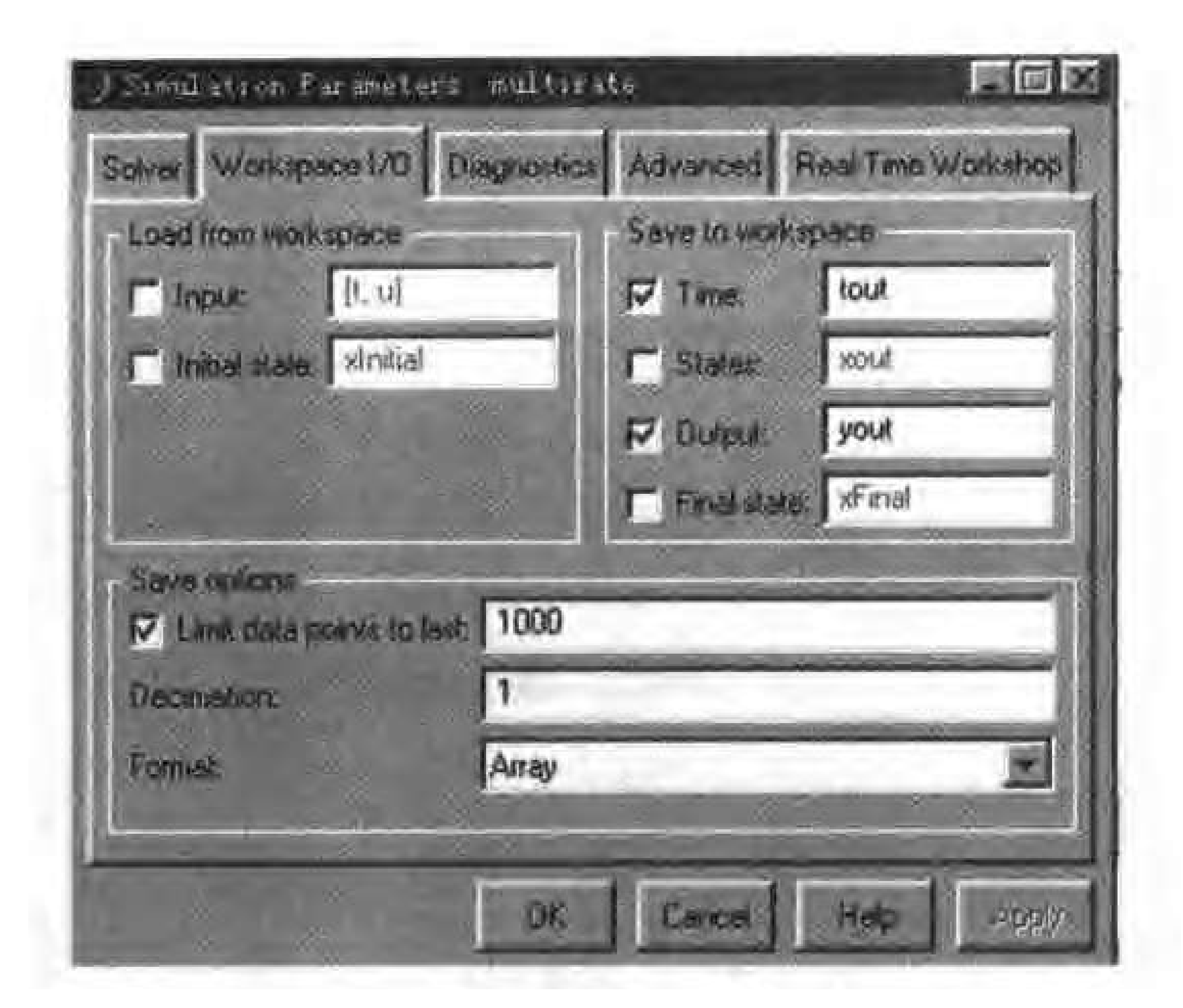


图 8-18 Workspace I/O 页设置

然后,请读者在 MATLAB 命令出口输入

```
>>set_param('exam_542','PreLoadFcn','ini_gain');
>>set_param('exam_542','StopFcn','out_graphic');
```

之后,保存模型,并关闭模型。打开 MATLAB 编辑器,新建一个名为 ini_gain 的 M 文件,并在里面输入

```
gain = 2;
```

再新建一个名为 out_graphic 的 M 文件,里面的语句为

```
plot(tout,yout);
```

保存这些 M 文件之后,打开 exam_542 模型并且运行它,读者可以发现,仿真结束之后,Simulink 会自动调用回调函数对保存在工作空间的输出变量进行处理,这里是用 plot 命令经模型输出的波形。

2. 建立动态对话框

这里用一个简单的例子,演示如何建立动态对话框。关于它的基本原理在前面章节已经介绍过了,这里就不再重复了。

请读者新建一个模型,并把一个 SubSystem 模块复制到模型中,请按照图 8-19 所示的对话框对子系统模块进行封装。封装对话框的示意图为图 8-20。

表 8-3 给出了其中各个模块参数的具体意义。

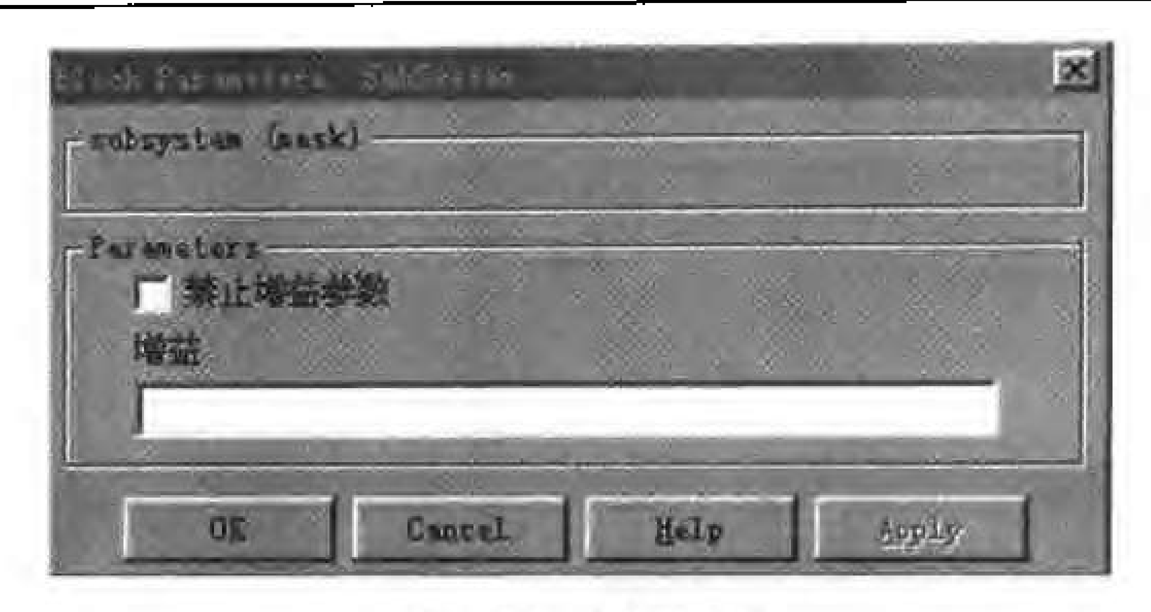


图 8-19 动态对话框

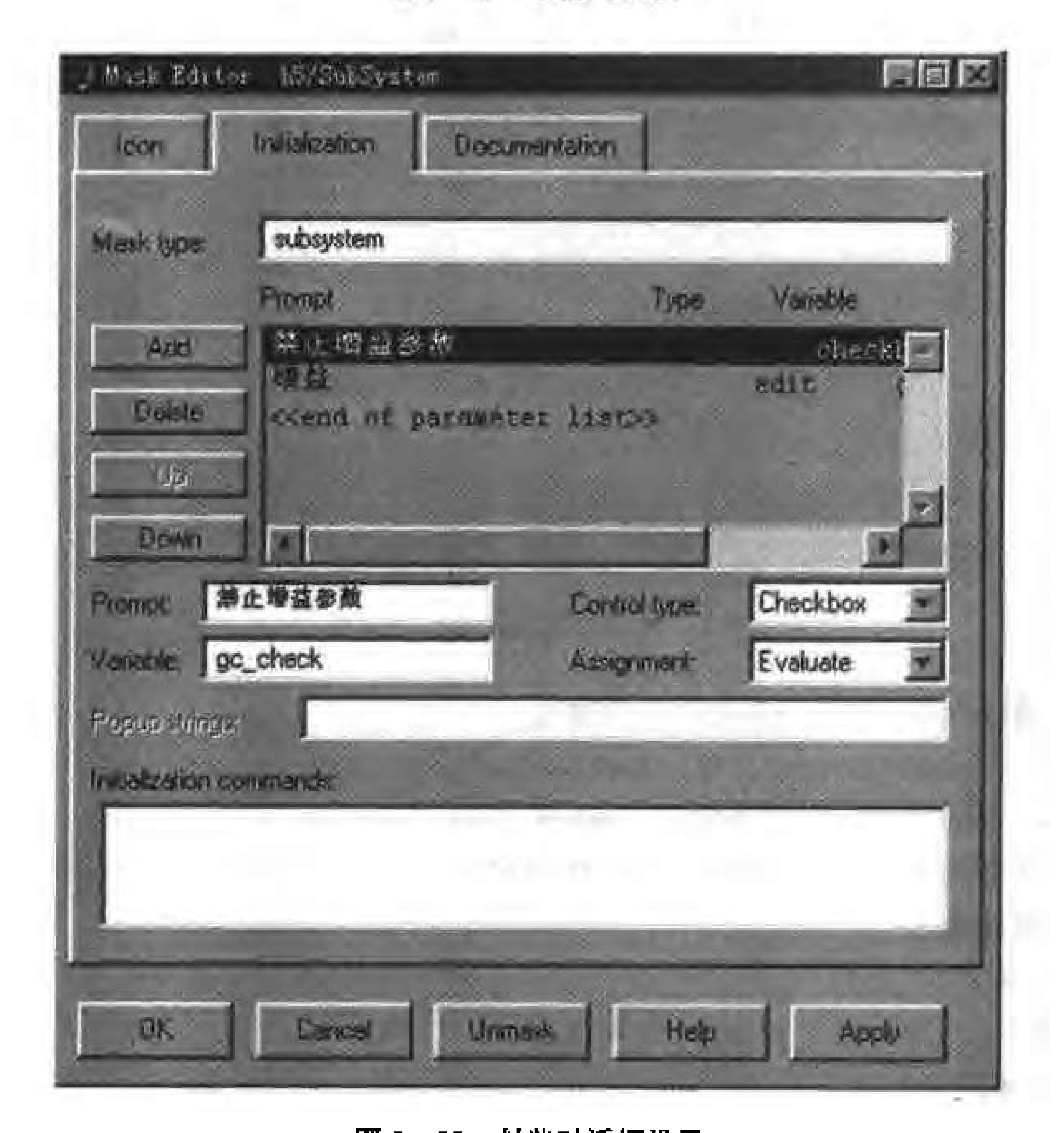


图 8-20 封装对话框设置

表 8-3 各个模块参数的具体定义

Prompt	Variable	Coutrol type	Assignment
禁止增益参数	g_ check	Chechbox	Evaluato
增益	Gain	Edit	Evaluate

这个例子要达到的效果是，当"禁止增益参数"检查框被选中之后，增益编辑框应该不

能输入，也就是处于禁止状态，呈现为灰色。下面就通过回调函数来实现这个功能。

为此，首先要为检查框控件定义回调函数。具体的操作，先在模型里选择子系统模块，然后在 MATLAB 工作空间输入下面的命令

```
> > set _ param (gcb,'MaskCallback', {'disable _ gain',});
```

这个命令通过设置当前模块(子系统模块)的第一个参数的回调函数为 disable _ gain，别忘了保存模型。于是第一个参数被改变时，Simulink 会自动调用这个 disable _ gain.m 文件。因此只要在 disable _ gain.m 里写入控制“增益”编辑框的语句即可。下面列出了 disable _ gain.m 的代码。

```
M _ value = get _ param (gcb, 'MaskValues') ; %获得参数控件的当前值
If strcmp (char(m _ value (1), 'on') %检查框的值是‘on’吗
set _ param (gcb, 'MaskEnables', {'on', 'off '} ; %是，将第二个参数控件禁止
else
set _ param (gcb, 'MaskEnables', {'on', 'on'} ; %否，将第二个参数控件使能
end
```

至于上面代码中的各个封装对话框参数的作用，请读者看前面章节。这里只想解释一下 if 判断语句。因为 m _ value 返回的是一个 cell 类，它不支持直接进行比较，所以先使用 char 函数将其转化为字符串，然后再用 strcmp 命令比较它是否和 on 相同。也许有的读者还会问，对于 checkbox 和它不是有一个变量关联，那为什么不用它来进行判断，直接是整数就不用这么麻烦。问题在于那里面的关联变量是在封装子空间定义的，而在 MATLAB 工作空间自然就无法访问了。

至此，整个动态对话框就完成了，可以双击观察完成后的动态对话框的效果。

8.5 模型文件格式

Simulink 为用户提供了图形界面和命令行两种方式来建立模型，设置模型的参数。这些方法，在某些场合给人的感觉是不太方便的。实际上，Simulink 允许用户直接修改模型文件，当然这要建立在对它的足够了解的基础上。这一节就粗略介绍 Simulink 的模型文件格式。

在 Simulink 里，每一个模型(包括库)都保存在一种以 .mdl 为后缀名的文件里，称为模型文件。概括地说，一个模型文件是一个结构化了的 MATLAB 文件，它包括描述模型的关键字和参数。读者可以通过 MATLAB Edit 来查看模块文件，方法和打开一般的 M 文件没什么区别，可以用 File 菜单下的 open 命令，但是打开时要把打开对话框的文件类型变成所有文件，否则就看不到当前目录中的模型文件，图 8 - 21 是一个模型文件(Simulink 的示例模型 vdp，其位置是在读者的安装目录下 toolbox \ Simulink \ simdemos)，显示在 MATLAB edit 中。

从上面的模型文件中，读者可以很明显地看出 Simulink 模型文件结构的层次性。

模型文件的结构如下：

```
Model {
< Model Parameter Name> < Model Parameter Value>
```

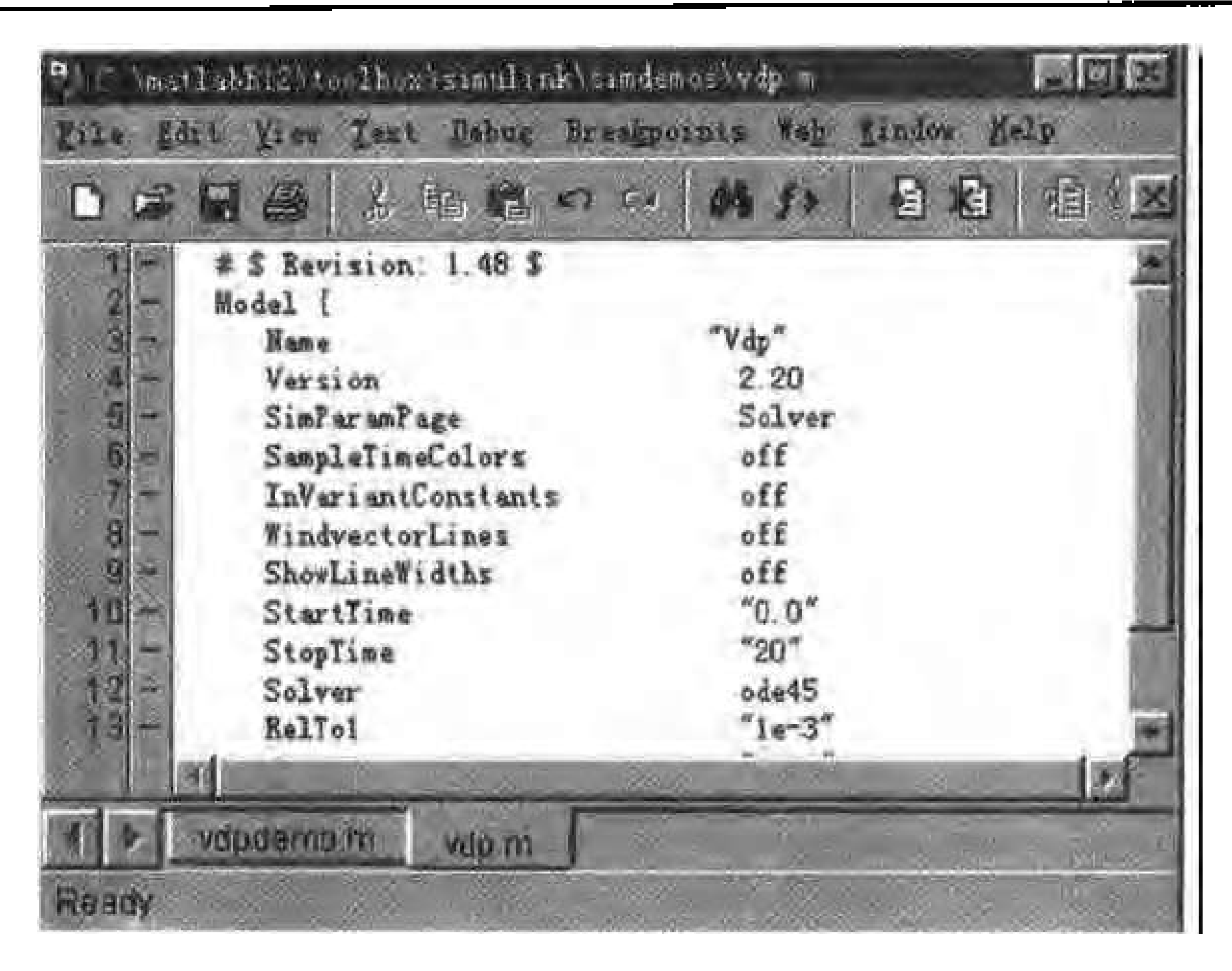


图 8-21 用 MATLAB 编辑器打开的模型文件

……

BlockDefaults {

<Block Parameter Name> <Block Parameter Value>

……

}

AnnotationDefaults {

<Annotation Parameter Name> <Annotation Parameter Value>

……

}

System {

<System Parameter Name> <System Parameter Value>

……

Block {

<Block Parameter Name> <Block Parameter Value>

……

}

Line {

<Line Parameter Name> <Line Parameter Value>

……

Branch {

<Branch Parameter Name> <Branch Parameter Value>

……

}

```
}
Annotation {
< Annotation Parameter Name > < Annotation Parameter Value >
......
}
}
}
```

读者可以对照这个基本结构来浏览 vdp 模型的模型文件。

Simulink 的模型文件是由描述的不同组成部分的 section 构成的,它们分别是:

(1)定义模型参数的 Model section ;

(2)包含模型中的模块的缺省设置的 BlockDefaults section ;

(3)包含模型中注解的缺省设置的 AnnontationDefaults section ;

(4)包含模型中的每一个系统(包含顶层系统和每一个子系统)的描述参数的 System section。每一个 System section 包含 block、line 和 annotation 等描述。

下面就来具体看看这些 section。为了说明方便,我们用第三章的封装子系统一节中的 mx + b 模型(如图 8 - 22)的模型文件为例来说明各个 section。

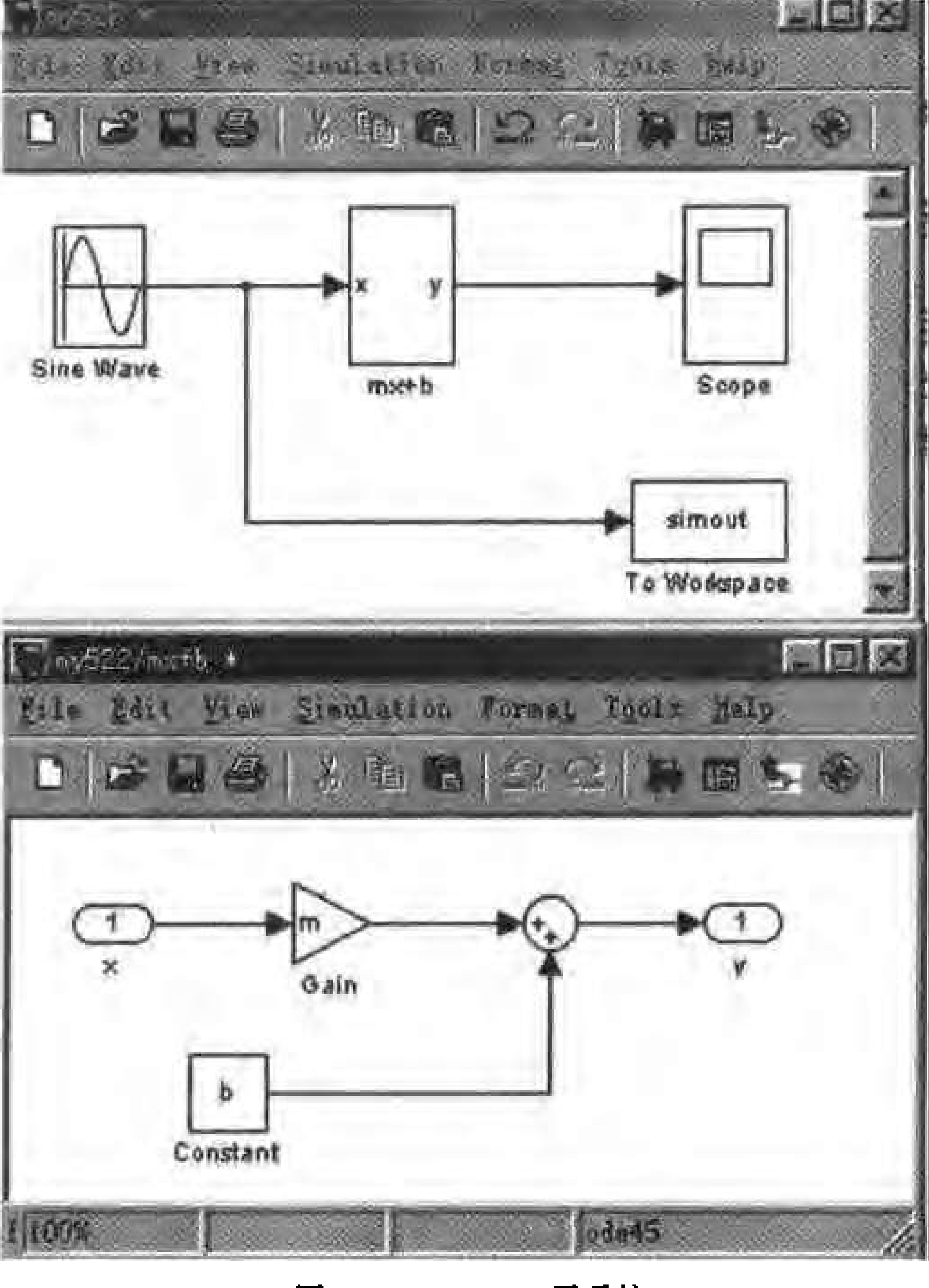


图 8 - 22 mx + b 子系统

(1)Model section。Model section 位于模型文件的顶部,它定义了模型层次参数的取值。这些参数包括模型名称、模型版本和仿真参数。读者可以在下面的示例模型文件的 Model section 里找到在本章里介绍的用仿真参数对话框设置的各种仿真参数。也就是说,读者完全可以从模型文件设置参数。

例如,下面代码所示的原来的模型文件中 LimitMaxRows 和 MaxRows 这两个参数的值分别是 off 和"1000"。如果读者把它们分别改为 on 和"2000",保存模型文件后(注意如果已经打开了该模型图表,请先关闭,关闭时请不要保存文件,否则会覆盖了在 matlab edit 中做的修改),再在 Simulink 里打开模型。查看该模型的仿真参数,读者会发现在对话框里的相应参数也做了修改。

```
Model {
Name                    "sampleformast"
Version                 3.00
SimParamPage            "Solver"
SampleTimeColors        off
InvariantConstants      off
WideVectorLines         off
ShowLineWidths          off
ShowPortDataType        off
StartTime               "0.0"
StopTime                "10.0"
SolveMode               "Auto"
Solver                  "ode45"
RelTol                  "1e-3"
AbsTol                  "auto"
Refine                  "1"
MaxStep                 "auto"
InitialStep             "auto"
FixedStep               "auto"
MaxOrder                5
OutputOption            "RefineOutTimes"
OutputTimes             "[]"
LoadExternalInput       off
ExternalInput           "[t, u]"
SaveTime                on
TimeSaveName            "tout"
SaveState               off
StateSaveName           "xout"
SaveOuoput              on
OutputSaveName          "yout"
```

LoadInitialState	off
Initialstate	"xInitial"
SaveFinalState	off
FinalStateName	"xFinal"
SaveFormat	"Matrix"
LimitMaxRows	off
MaxRows	"1000"
Decimation	"1"
AlgebraicLoopMsg	"warning"
MinStepSizeMsg	"warning"
UnconnectedInputMsg	"waring"
UnconnectedOutputMsg	"waring"
UnconnectedLIneMsg	"waring"
InheritedTsInSrcMsg	"waring"
IntegerOverflowMsg	"waring"
UnneccessaryDatetypeConvMsg	"none"
Int32ToFloatConvMsg	"waring"
SignalLabelMismatchMsg	"none"
ConsistencyChecking	"off"
ZeroCross	on
SimuliationMode	"normal"
BlockDataTips	on
BlockParametersDataTip	off
BlockAttributesDataTip	off
BlockPortWidthsDataTip	off
BlockDescriptionStringDataTip	off
BlockMaskParametersDataTip	off
ToolBar	off
StatusBar	off
BrowserShowLibraryLinks	off
BrowserLookUnderMasks	off
OptimizeBlockIOStorage	on
BufferReuse	on
BooleanDateType	off
RTSystemTargetFile	"trt.tlc"
RTWInlineParameters	off
RTWRetainRTWFile	off
RTWTemplateMakefile	"grt_default_tmf"
RTWMakeCommand	"make_rtw"

```
RTWGenerateCodeOnly          off
ExtModeMexFile               "ext_comm"
ExtModeBatchMode             off
ExtModeTrigType              "manual"
ExtModeTrigMode              "oneshot"
ExtModeTrigPort              "1"
ExtModeTrigElement           "any"
ExtMOdeTrigDuration          1000
ExtMOdeTrigHOldOff           0
ExtModeTrigDelay             0
ExtMOdeTrigDirection         "rising"
ExtMOdeTrigLevel             0
ExtMOdeArchiveMode           off
ExtMOdeAutoIncOneShot        off
ExtModeIncDirWhenArm         off
ExtMOdeAddSuffixToVar        off
ExtMOdeWriteAllDataToWs      off
ExtModeArmWhenConnect        off
Created                      "Tue Dec 12 21 : 40 :29 2000"
UpdateHistory                "UpdateHIstoryNever"
ModifiedByFormat             "% < Auto > "
ModeifiedDateFormat          "% < Auto > "
LastModifiedDate             "Fri Dec 29 16 : 19 : 33 2000"
ModelVersionFormat           "1.% < AutoIncrement : 7 > "
ConfigurationManager         "none"
}
```

（2）BlockDefaults section。BlockDefaults section 出现在模型文件的仿真参数后面，它定义了这个模型内的模块参数的缺省值。这些值可以被 block section 里定义的单个模块参数重定义。

```
BlockDefaults {
Orientation                  "right"
ForegroundColor              "black"
BackgroundColor              "white"
DropShadow                   off
NamePlacement                "normal"
FontName                     "Helvetica"
FontSize                     10
FontWeight                   "normal"
FontAngle                    "normal"
```

```
ShowName                    on
}
```

(3) AnnotationDefaults section。它出现在 BlockDefaults section 后面，定义了模型中所有注释的参数的缺省值。这些参数值不能用 set _ param 命令来修改。

```
AnnotaitonDefaults {
  ForizontalAlignment       "center"
  VerticalAlignment         "middle"
  ForegroundColor           "black"
  BackgroundColor           "white"
  DropShadow                off
  FontNme                   "Helvetica"
  FontSize                  10
  FontWeight                "normal'
  FontAngle                 "normal"
}
LineDefaults {
  FontName                  "Helvetica'
  FontSize                  9
  FontWeight                "normal"
  FontAngle                 "normal"
}
```

(4) System section。顶层的系统和模型中的每一个子系统都在一个单独的 System section 里描述。每一个 System section 部分定义了系统级的参数，并包含系统内每一个模块的 Block、Line 和 Annotation 节，以及系统自身的 Line 和 Annotation 小节。具有分枝点的直线还包含一个 Branch section 来定义分枝直线。

这里，读者仔细体会模型文件体现出的模型结构的层次性。在 System section 里有四个 Block section，分别对应四个顶层模块：scope、sin、toWorkspace 和子系统模块 mx + b。而子系统 mx + b 对应的 Block section 又嵌套了一个 System section，用以说明子系统内部的模块和连线的参数。

```
System {
Name                        "sampleformask"
Location                    [160,123,500,260]
Open                        on
ModelBrowserVisibility      off
ModelBrowserWidth           200
ScreenColor                 "automatic"
PaperOrientation            "landscape"
PaperPositionMode           "auto"
PaperType                   "usletter"
```

```
PaperUnits              "inches"
ZoomFactor              "100'
AutoZoom                on
ReportName              "Simulink - default.rpt"
Block {
   BlockType            Scope
   Name                 "scope"
   Ports                [1,0,0,0,]
   Position             [240,19.270,51]
   Floating             off
   Location             [305,135,629,375]
   Open                 off
   NumInputPorts        "1"
   TickLabels           "OneTimeTick"
   ZoomMode             "on"
   List {
ListType                AxesTitles
Axesl                   "% < SignalLabel > "
   }
   Grid                 "on"
   TimeRange            "auto"
   YMin                 " - 5"
   YMax                 "5"
   SaveToWorkspace      off
   SaveName             "ScopeData"
   DateFormat           "StructureWithTime"
   LimitMaxRows         on
   MaxRows              "5000"
   Decimation           "1"
   SampleInput          off
   SampleTime           "0"
}
Block {
   BlockType            Sin
   Name                 "Sine Wave"
   Position             [50, 20, 80,50]
   Amplitude            "1"
   Frequency            "1"
   Phase                "0"
```

```
    SampleTime              "0"
  }
  Block {
    BlockType               ToWorkspace
    Name                    "To Workspace"
    Position                [225,85,285,115]
    VariableName            "simout"
    Buffer                  "inf"
    Decimation              "1"
    SampleTime              "-1"
    SaveFormat              "Structure"
  }
  Block {
    BlockType               SubSystem
    Name                    "mx + b"
    Ports                   [1,1,0,0,0]
    Position                [135,12,175,58]
    ShowPortLables          on
    MaskType                "mx + b"
    MaskDescription         "这个模块求出 y = mx + b"
    MaskPromptString        "常数项:/增益系数"
    MaskStyleString         "edit, edit"
    MaskTunableValueString  "on, on"
    MaskCallbackString      "1"
    MaskEnableString        "on, on"
    MaskVisibilityString    "on, on"
    MaskVariables           m = @1; b = @2;
    MaskDisplay R           " \ nplot([0 1], [0 m] + (m < 0); \ ndisp('mx +
                            b');"" \ n \ n \ n \ n"
    MaskIconFrame           on
    MaskIconOpaque          on
    MaskIconRotate          "port"
    MaskIconUnits           "normalized"
    MaskValueString         "210"
    System {
Name                        "mx + b"
    Location                [282,214,565,342]
    Open                    off
    ModelBrowservisibility  off
```

```
    ModelBrowserWidth       200
    ScreenColor             "white"
    PaperOrientation        "landscape"
    PaperPositionMode       "auto"
    PaperType               "usletter"
    PaperUnits              "inches"
    ZoomFactor              "100"
    AutoZoom                on
    Block {
      BlockType             Inport
      Name                  "x"
      Position              [15,23,45,37]
      Port                  "1"
      PortWidth             "-1"
      SampleTime            "-1"
      DataType              "auto"
      SignalType            "auto"
      Interpolate           on
}
    Block {
      BlockType             Constant
      Name                  "Constant"
      Position              [45,65,75,95]
      Value                 "b"
}
    Block {
      BlockType             Gain
      Name                  "Gain"
      Position              [80,15,110,45]
      Gain                  "m"
      SaturateOnIntegerOverflow on
}
    Block {
      BlockType             Sum
      Name                  "Sum"
      Ports                 [2,1,0,0,0]
      Psition               [140,20,160,40]
      ShowName              off
      IconShape             "round"
```

```
      Inputs                        "1 + + "
      SaturateOnIntegerOverflow     on
    }
    Block {
      BlockType                     Outport
      Name                          "y"
      Position                      [200,23,230,37]
      Port                          "1"
      OutputWhenDisabled            "held"
      InitialOutput                 "[]"
    }
    Line {
      SrcBlock                      "x"
      SrcPort                       1
      DstBlock                      "Gain"
      DstPort                       1
    }
    Line {
      SrcBlock                      "Gain"
      SrcPort                       1
      DstBlock                      "Sum"
      DstPort                       1
    }
    Line {
      SrcBlock                      "Constan"
      SrcPort                       1
      Points                        [70,0]
      DstBlock                      "Sum"
      DstPort                       2
    }
    Line {
      SrcBlock                      "Sum"
      SrcPort                       1
      DstBlock                      "y"
      DstPort                       1
    }
      }
    }
    Line {
```

```
      SrcBlock                "Sine Wave"
      SrcPort                 1
      Points                  [20,0]
      Branch {
  DstBlock                    "mx + b"
  DstPort                     1
      }
      Branch {
  Points                      [0,65]
  DstBlock                    "To Workspace"
  DstPort                     1
      }
    }
    Line {
      SrcBlock                "mx + b"
      SrcPort                 1
      DstBlock                "Scope"
      DstPort                 1
      }
    }
}
```

第 9 章　使用 Real - Time Workshop

9.1　Real - Time Workshop 概述

9.1.1　Real - Time Workshop 能做什么

在进入具体的学习之前，读者应该先弄清楚 Real - Time Workshop(以后简称为 RTW)能做些什么，对自己的工作是否有帮助。

简而言之，RTW 是和 MATLAB、Simulink 一起使用的一个工具，它可以直接从 Simulink 模型生成代码并且自动建立可以在不同环境下运行的程序，这些环境包括实时系统和单机仿真。单机仿真的概念指是，用户可以不用打开 MATLAB 和 Simulink 模型图表就可以运行仿真，因为 RTW 根据模型建立了一个可执行的程序。

RTW 提供的另外一个很有诱惑力的功能是，通过它，用户可以在远程处理器实时地运行 Simulink 模型，并且它所生成的单击仿真程序同样具备外部计算机运行的能力。

RTW 能够应用的场合是十分广泛的，下面列举了它的几个例子。

(1)实时控制——可以使用 MATLAB 和 Simulink 设计控制系统，并且从建立的模型图表模型生成代码。可以编译它们和载入它们到目标硬件。

(2)实时信号处理——可以使用 MATLAB 和 Simulini 设计信号处理算法，同样可以从模型生成代码，编译和载入它们到目标硬件。

(3)硬件环路仿真——可以建立模仿实际生活测量、系统动力和激励信号的 Simulink 模型。从模型生成的代码可以被定位到特殊用途的硬件(如 DSP)，它提供了物理系统的实时表示。

(4)交互的实时参数调整——使用 Simulink 作为建立的实时程序的前端，就可以利用 Simulink 的图形界面，在程序执行时改变参数。

(5)高速的单击仿真。

(6)生成可插入到其他仿真程序的便携 C 代码。

无论是何种应用，代码生成都是其中必需的过程。RTW 生成的代码在缺省情况下是高度优化和完全注释的 C 代码。从任何 Simulink 模型都可以生成代码，它们包括：线性、非线性、连续、离散以及混合模型。

所有的 Simulink 模块都自动地转化为代码，除了 MATLAB function 模块和调用 M 文件 S - 函数的 S - function 模块。如果想把它们和 RTW 一起使用，就需要重写这些模块为 C MEX S - 函数。

RTW 包含一系列的目标文件，并由目标语言编译器(TLC)编译生成 ANSI C 代码。

目标文件是ASCII文件,描述如何将Simulink模型转化为代码。对于高级用户,目标文件使得它能够自定义生成代码。

读者还可以把C MEX S-函数和生成的代码一起合并到可执行程序。同样还可以为自己的C MEX S-函数编写一个目标文件库里内嵌S-函数,这样就可以通过减少对S-函数的调用来提高性能。

RTW可以输出的类型有:

(1)C代码——为Simulink模型生成包含系统方程和初始化函数的代码,对这可以在非实时环境或者实时环境使用这些代码。

(2)Ada代码——从SIMULINK模块生成Ada代码,这要求用户安装Real-Time Workshop Ada Coder。

(3)实时程序——将代码转换为适合指定实时硬件运行的实时程序。对应的代码被设置为和一个外部时钟源相连接,并且以一个固定的由用户设定的采样速率运行。

(4)高性能的单击仿真——将生成代码和普通实时系统目标文件一起使用,为单机仿真生成可以执行的程序。在仿真结束时,可执行文件产生一个model.mat文件,它里面包含了Simulink保存仿真数据的MATLAB变量,这个文件可以在MATLAB里进行分析。

在大致了解RTW的主要用途之后,读者如果对RTW的使用有兴趣,那么就请您继续阅读本章的内容。RTW的内容是相当多的,本章只介绍它的一些比较基本的使用,关于更详细的说明,可以参阅MATLAB的帮助文件。

9.1.2　使用前的准备工作

在读者确信RTW对自己的工作有所帮助之后,那么请在进入学习之前,先做一些准备工作,这主要是一些软件的安装和选项的简单设置。

首先,读者当然要在所使用的机器上安装好RTW,这一点使用MATLAB安装向导是很容易办到的。

接着,读者要安装一些第三方编译器,这些编译器是在读者使用RTW生成可执行程序时,供RTW访问调用的。为此,还要进行一定的设置。RTW支持的几种第三方编译器有:

Watcom——支持的版本有:10.6,11.0;

Borland——支持的版本有:5.0,5.2,5.3;

Microsoft Visual C/C++——支持的版本有:4.2,5.0,6.0。

读者可以选择其中的一种安装,并且按照下面的描述进行设置。首先,要设置的是环境变量,它要求正确的指向编译器所在的位置。

(1)Watcom。定义WATCOM环境变量,在Windows95或者98下,读者可以把下面的命令添加到autoexec.bat文件,这个文件在C:\Windows目录下。

Set WATCOM = <Watcom编译器的安装路径>

例如,set WATCOM = D:\Watcom。

可以通过在dos提示符下输入

set WATCOM

来检查WATCOM环境变量是否定义,并且正确地指向Watcom编译器所在的目录。

这个命令将返回所选择的目录。

在 Windows NT 下,可以在控制面板选择 System,进入 Environment 页,然后定义 WATCOM 变量,并且输入编译器的路径。

(2)Microsoft Visual C/C + +。对于 Microsoft Visual C/C + + 编译器,则要定义环境变量 MSDevDir 为:

MSDevDir = <编译器的路径> for Visual C/C + + 4.2

MSDevDir = <编译器的路径> \ SharedIDE for Visual C/C + + 6.0

MSDevDir = <编译器的路径> \ Common \ MSDev98 for Visual C/C + + 6.0

例如,set msdevdir = c: \ program files \ Microsoft visual studio \ common \ msdev98。

在设置过程中,如果遇到"out - of - environment space"这样的错误信息,读者可以用鼠标右键单击产生这个问题的程序(例如,autoexec.bat),并且从弹出的菜单中选择 Properties 命令,从打开的对话框中选择 Memory 页面,并把 Initial Environment 属性设为所能允许的最大值,然后按 Apply 按钮应用上的改动。

其次,还要进行的一项设置是修改 MATLAB 的搜索路径。因为 RTW 在建立程序时,会调用第三方的编译器的 make 程序,为了让 RTW 能够找到 make 程序,读者要把 make 程序所在的目录添加到 MATLAB 的搜索路径中。

9.1.3 RTW 中的基本概念

在一步接一步地学习使用 RTW 的一个个例子之前,读者先必须了解 RTW 中的一些基本概念。这些概念包括:

(1)普通定时;

(2)目标定位;

(3)目标语言编译文件;

(4)模板 makefile;

(5)建立过程;

(6)配置模板文件;

(7)设定仿真参数。

1. 普通定时

RTW 提供了普通的实时发展对象,所谓普通实时是:

(1)一个在单任务或者多任务模式下仿真固定步长模型环境;

(2)实现代码验证的一种方法。

2. 目标定位

使用 RTW,读者必须决定放置生成代码运行的环境,包括硬件或者操作系统,这称为目标定位,环境本身就成为目标(target)。而宿主(host)就是用户运行 MATLAB、Simulink 和 RTW 的系统。使用 RTW,可以生成代码以及能够在目标系统中的可执行程序。生成特定目标的代码的过程由系统目标文件、模板 makefile 文件和 make 命令来控制。系统目标文件和模板 makefile 文件定义了目标的类型,它们必须在仿真参数对话框的 Real - Time Workshop 页被设定。

3. 目标语言编译文件

目标语言编译文件(Traget Language Compiler files 或 TLC),是供目标编译器编译和执行的文件,它描述了如何将 Simulink 模型翻译为用户所设定的目标代码。RTW 使用 TLC 文件来把 Simulink 模型翻译成代码,而系统目标文件是 TLC 程序生成可执行程序的进入点。

4. 建立过程

RTW 建立过程由 make _ rtw 来控制,这个程序在用户点击 Real - Time Workshop 页上的 Build 按钮(见后面的小节)后被调用。首先,make _ rtw 编译模块图表并且生成一个 model.rtw 文件。然后,make _ rtw 从 Real - Time Workshop 页上指定的模板 makefile 生成一个 makefile,名为 model.mk。最后,如果运行的 host 和模板 makefile 的 HOST 宏匹配,那么就调用 make 程序,从而生成代码建立程序。

5. 模板 makefiles

RTW 使用模板 makefile 生成代码建立可执行程序。按照惯例,一个模板 makefile 有一个 .tmf 的后缀名,还有一个和所应用目标相对应的名称。例如,grt _ unix.tmf 是用于 UNIX 的普通实时模板 makefile,这个模板文件应用的目标是 grt _ unix。

从模板 makefile 生成的 makefile 逐行地复制模板 makefile 的每一行,并且把记号扩展到 makefile。生成的 makefile 名为 model.mk,它被传给 make 命令,make 再根据一些文件来生成可执行程序。

6. 模板 makefile 配置

读者可以巩固修改模板 makefile 来配置建立过程。完成的过程可以是,把模板文件复制到你的当前工作目录,并且编辑它。或者可以通过在 make _ rtw 命令后面添加选项来配置模板 makefile。例如,为 grt _ unix.tmf 打开调试符,读者可以在仿真参数对话框设定建立命令为:

make _ rtw OPT _ OPTS = -g %OPT _ OPTS = -g 就是添加的选项设置

7. 设定仿真参数

读者知道,修改仿真参数可以控制仿真的行为,例如起始和终止时间,关于它们的设置,本书的前面章节已经介绍得很清楚了,这里就不说了。请记住,仿真参数直接影响到代码生成和程序建立,所以,在生成代码和建立程序之前,读者必须首先验证参数是否在仿真对话框设置正确。

9.2 生成普通的实时程序

这个例子演示如何从一个 Simulink 模型生成 C 代码和普通的实时程序,这个程序有以下的特点:

(1)它作为一个单机程序执行,独立于外部的定时和事件。

(2)它保存数据在 MATLAB MAT 文件里作为后来的分析用。

(3)它是在 unix 或者 pc 环境下建立的。

这个示例程序同样演示如何使用数据调入来验证生成的代码。数据保存(data logging)允许用户将从 Simulink 模块图表得到的系统输出和由生成代码产生的程序得到的

数据进行比较。

9.2.1 Simulink 模型

这个例子使用 Simulink 的示例模型(在 simdemos 目录里)f14 作为生成代码的源模型。f14 是一个模拟控制 f14 飞机纵向运动的模型。在这里,它的工作原来不是讨论的重点。

在讲解之前,先对后面将要用到的名称进行约定:在 MATLAB 命令行输入 f14 表示打开 f14 模型,这和本书前面所讲到的约定是一致的。输入 f14 的作用是执行名称为 f14 的文件,"!"就相当于把命令转到操作系统去。而 load f14 这是从 f14.mat 文件中读取数据变量 f14,最后,F - 14 代表了飞机。

请在 MATLAB 命令行输入 f14 来显示这个模型的图。

```
>>f14
```

图 9 - 1 就是这个模型的模块图。

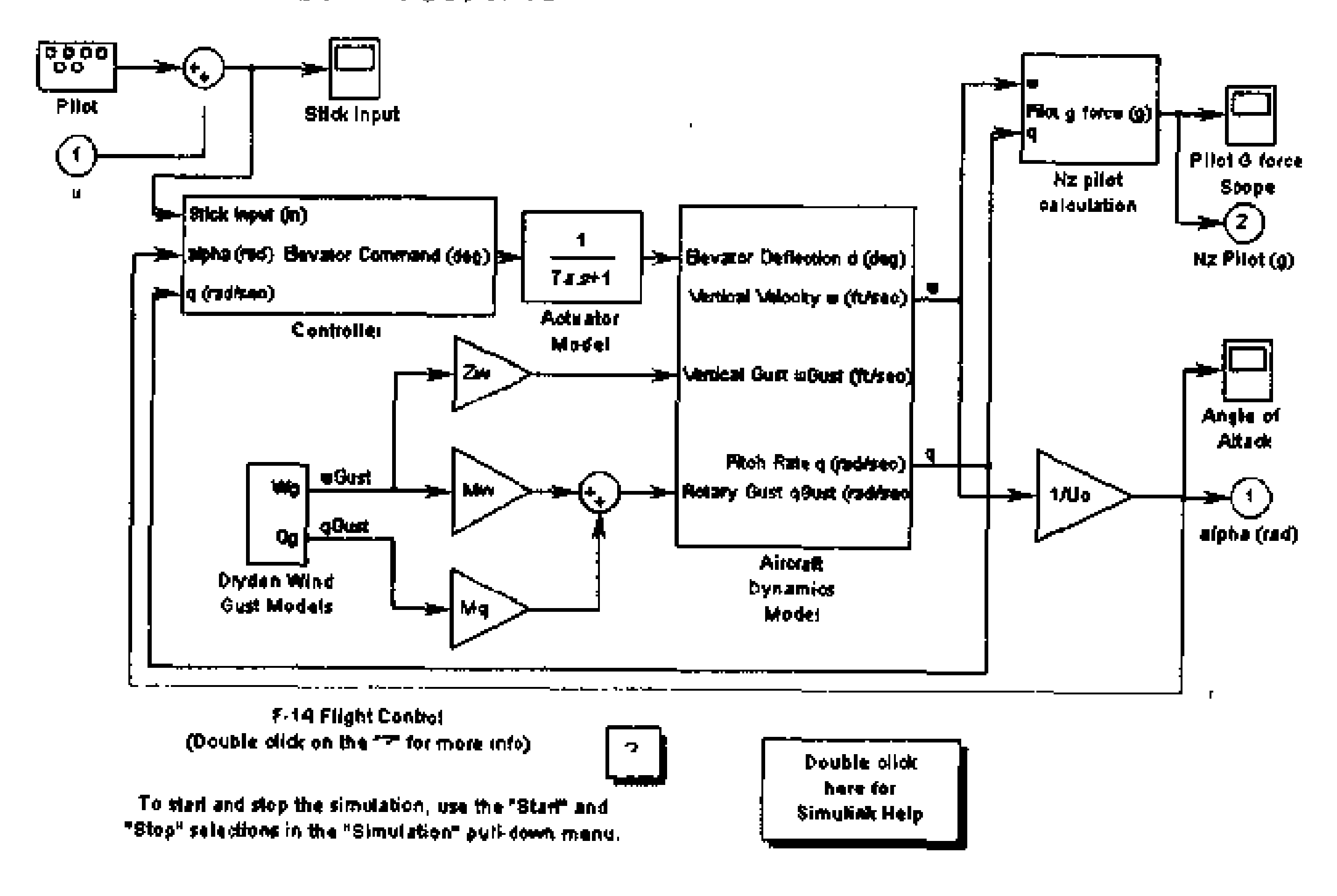


图 9 - 1 f14 的模型图

在介绍如何生成代码之前，让我们先来粗略地审视一下 f14。读者可以看到，f14 模型有四个比较熟悉的模块：一个信号发生模块和三个 Scope 模块。其中，信号发生模块用来模拟飞行员的操作输入，它的波形由一个 Scope 模块来跟踪，而剩余的 Scope 模块则用来跟踪系统的输出——攻击的飞行角度和飞行员所承受的重力，使用 Scope 模块显示波形是 Simulink 里跟踪模型行为的常用方法。至于模型中的其他模块（系统），读者可以自己一层层地打开来查看。它们涉及到 f14 模型的设计和内部工作原理，读者可能会对这些很迷惑，但是也不必担心。这个示例的目的只是让读者知道如何从 Simulink 模型生成普通的实时代码，而生成代码所用到的操作都是和模型的内容和复杂性无关的。

需要强调的是,读者可以使用别的 Simulink 模型来生成实时代码。

9.2.2 生成实时代码

1. 设置程序的参数

生成实时代码首先要进行的操作是设置一些参数。在打开 f14 模型之后，请读者从下拉菜单 simulation 里选择 parameters 命令，打开仿真参数对话框。图 9－2 显示了 real－time workshop 页的情况，该页包含的操作将在本章详细介绍。

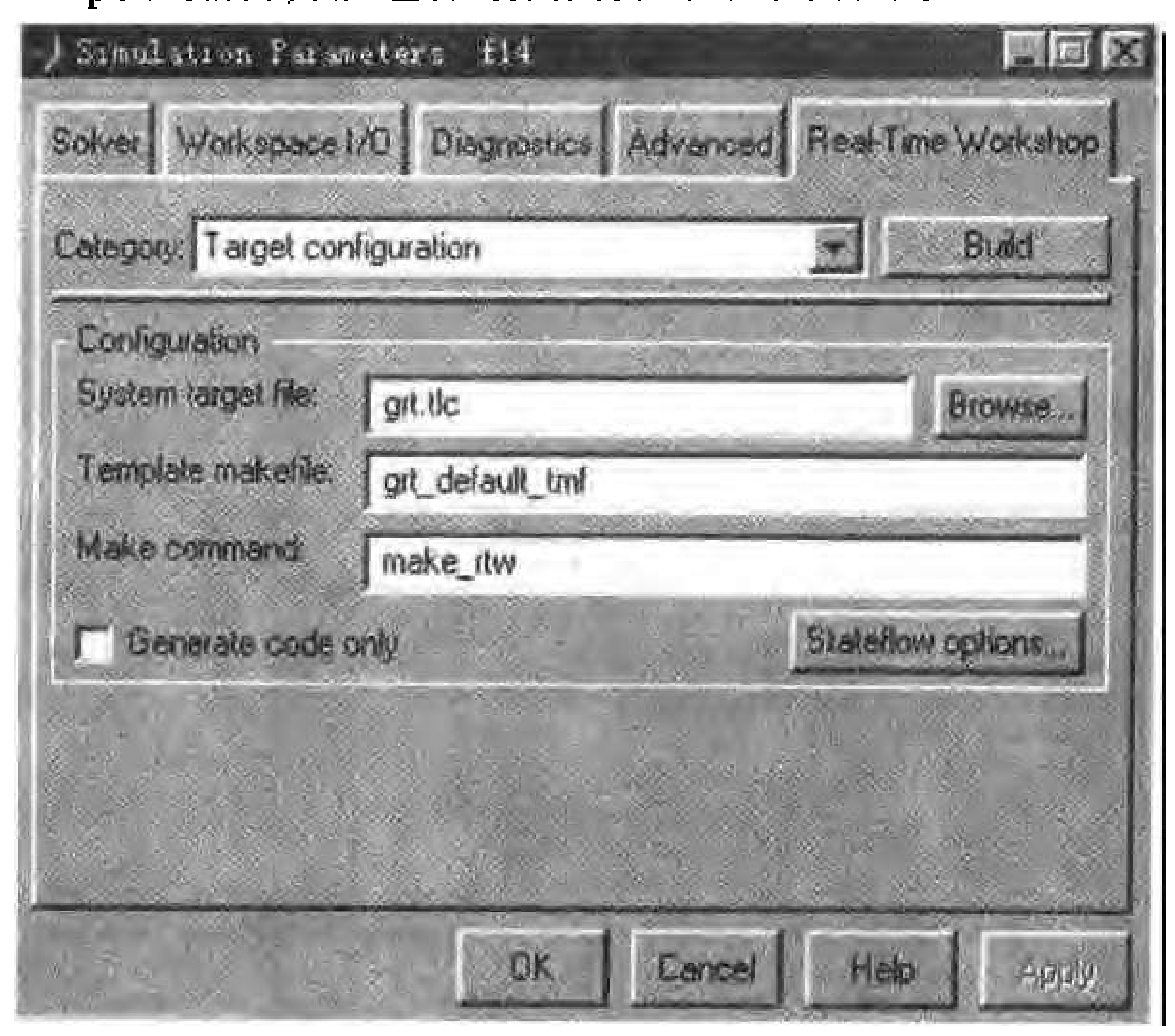


图 9－2 real－time workshop 页

Real－Time workshop 页是用户设置参数选项，选择模板制作文件(makefile)，产生代码和建立程序的地方。在对话框里初始显示的缺省参数值不一定会和用来生成可实时执行的代码所要求的参数匹配。因此，Simulink 要求用户在自己的模型使用 Real－Time Workshop 时，改变缺省参数来和模型的仿真参数匹配。要为 f14 模型进行配置，请读者按照下面所述来进行：

(1)在 Solver 页，设置 Solver options Type 参数为 Fixed－step，并且选择 ode5(Dormand－Prince)解法器。

(2)设置 Fixod Step Size 参数为 0.05。

设置完参数之后，就可以进入普通实时程序的环节了。

打开 Real－Time Workshop 页的方式，从 tools 下拉菜单选择 Real－Time。

2. 建立程序

要建立程序，请读者在仿真参数对话框里选择 Real－Time Workshop 页，然后按 Build 按钮就可以从 f14 模型生成 C 代码。Build 命令调用一个系统目标文件 grt.tlc，它使用指定的模板文件 rt _ default _ tmf，来生成 makefile。Real－Time Workshop 再使用生成的 makefile 来建立程序。

当对话框的 Build 按钮被按下之后，Real - Time Workshop 按下面的顺序来调用 make 命令：

(1) 编译模型图表生成 model.rtw 文件(在本例中，是 f14,rtw)。

(2)调用目标语言调用器，依次编译 TLC 程序，从 grt.tlc 开始，处理 model.rtw 生成代码。

(3)从模板 makefile 文件(例如，grt_vc.tmf)建立一个名为 model.mk 的 makefile。

(4)如果 Simulink 在和模板 makefile 文件所指定的相同宿主机上运行，那么实时程序就生成好了，否则，在生成代码后处理就中断，除非用户在模板 makefile 定义和目标相匹配的宿主机。

在模板 makefile 文件里定义的变量 HOST 用以识别用户的目标系统。HOST 的取值有三个选项：PC、UNIX 和 ANY。其中，ANY 的作用是让用户的目标系统不是所运行的这个而是其它(例如目标系统是 DSP 或者微处理器等等)。

一旦 Build 命令被执行，Real - Time Workshop 缺省情况下就生成下列文件：

(1)f14.c——单机 C 代码。

(2)f14.h——包含状态变量信息的包含头文件。

(3)f14_export.h——包含输出数据和参数的包含头文件。

(4)f14.reg——一个包含头文件，它包含了完成在生成代码中对数据结构进行初始化的模型注册函数信息。

(5)f14.prm——一个包含模型 f14 中使用的参数的有关信息的包含头文件。

(6)f14(UNIX)或者 f14.exe(PC 机上)——普通的实时可执行代码。

下面的执行结果就是在 PC 机上(操作系统为 Windows)执行 build 命令后，MATLAB 命令窗口显示的信息。当然显示在 MATLAB 命令窗口的信息和你的仿真参数设置有关。

```
### Starting Real - Time Workshop build procedure for model : f14
### Invoking Target Language Compiler on f14.rtw
### f14.mk which is generated from D: \ MATLABR11 \ rtw \ c \ grt \ grt_vc.tmf is up to date
### Building f14 : f14.bat
### Successful completion of Real - Time Workshop build procedure for model : f14
```

3. 自定义 Build 的过程

Simulink 还允许用户可以选中 Inline parameters 检查框来选中内置参数。这会导致 Real - Time Workshop 用模型参数值的取值替换变量名，来建立普通的实时程序。它同样指 Simulink 传递常数采样时间，这样可以把常数运算放入启动代码，改善代码的性能。Simulink 同样允许用户自定义模板 makefile 和 make 命令；这些选项将在本章的后面讨论。

单击 Generate code only 告诉 Real - Time Workshop 只产生代码而不编译，意味着目标和可执行文件没有生成。选择 Retain.rtw 将使 Real - Time Workshop 保存 model.rtw 文件。缺省情况下，这个文件在 Build 过程中被删除。

4. 数据记录

这个例子使用 Real - Time Workshop 的 MAT - file 数据记录功能来保存系统状态、输出以及仿真时间。为此，请在 Workspace I/O 页选中 Time、States 和 Outputs 选项使 Real - Time Workshop 记录仿真时间、任何离散或者连续状态和任何根输出端口模块。图 9 - 3 是 Workspace I/O 页的样子。

图 9 - 3 Workspace I/O 页

例如，可以选中 Time and Outputs 选项来设置数据记录变量，此外，Real - Time Workshop 也会把所有设置了保存数据到工作空间属性的所有 scope 模块的输入数据保存到一个 MATLAB MAT 文件。于是，用户就可以载入数据到 MATLAB 作分析。

具体操作是选中 scope 模块的特性设置对话框的数据历史(Data history)页的 Save data to workspace(保存数据到工作空间)选项，然后再在下面的编辑框里输入存储数据的变量名称。图 9 - 4 所示的是 Stick Input 模块的对话框，读者可以照此来设置其他的两个 scope 模块。

如此设置之后，就可以重新用 Build 命令来生成可执行代码，然后读者在命令窗口输入如下的命令：

```
>>! f14                     %执行生成可执行的程序

**starting the model**
**created f14.mat**
>>clear                     %清除以前在 MATLAB 工作空间已有的变量，以免混淆
>>load f14
```

```
>>who
Your variables are :
rt_Angle_of_attack
rt_pilot_G_force
rt_stick_input
```

f14.mat 文件里包含了可执行程序执行之后生成的几个输出数据变量,读者可以发现变量的名称和设置的变量不一样,它们都多了一个前缀 rt_,"rt_"是 real time 的简写,在后面读者会知道这个前缀是可以设置的。

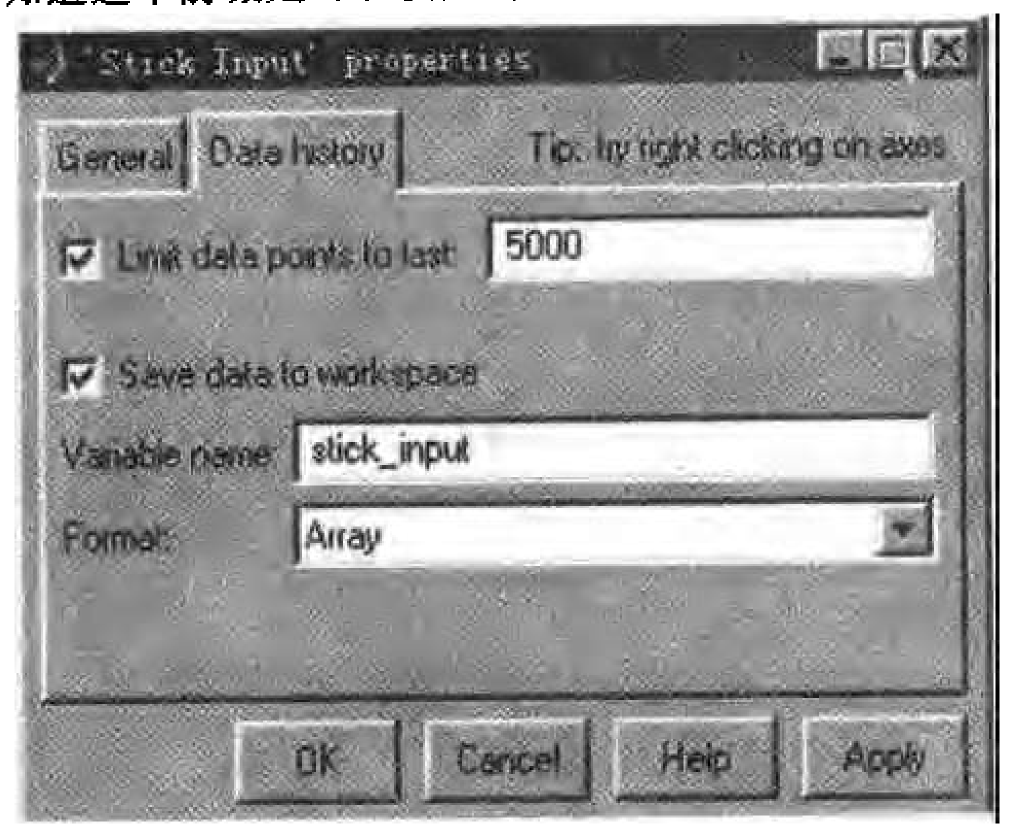


图 9-4 设置 scope 模块

为使实时代码具有数据记录能力,可以通过设置 Workspace I/O 页的保存数据到工作空间属性,也可以通过 scope 模块的属性对话框来设置。这两种方式是互不影响的。例如,读者可以只用 Workspace I/O 页来设置,例如选择时间和输出(请详见端口 1 和端口 2 的输出以及时间值保存到 mat 文件)。

5. 实时运行的接口

建立实时程序除了要根据模型生成代码之外,还需要一系列的源文件。这些源文件包括:

(1)主程序;

(2)驱动模块执行的代码;

(3)实现积分算法的代码;

(4)生成 SimStruct 数据结构的代码,SimStruct 用于管理模型的执行。

6. 设置模板 madefile

这个例子运用了两个不同的模板 makefiles:用 Unix 的 grt_unix.tmf,用于 PC 平台的 grt_vc.tmf、grt_wat.tmf 和 orgrt_bc.tmf(取决于所选择的编译器)。这些 makefile 模板自动处理设置好的 makefiles 来建立程序。用户可以把模板 makefile 复制到用户的当前目录,然后修改它,来修改建立可执行程序的过程。

这些文件的位置位于 matlab/rtw/c/grt 目录,读者可以用 MATLAB 的 matlabroot 命

令决定自己所用系统的 MATLAB 路径。

关于 Unix 模板 makefile,这里就忽略不讲,请使用 Unix 环境的读者查阅 RTW 帮助文档。主要来介绍面向 PC 的 makefiles。其中 grt_vc.tmf 和 grt_msvc.tmf 是为使用 Microsoft Visual C/C++编译器的用户设计的。要使用这些模板 makefiles 的任何一个,用户必须确保指向 Visual C/C++的环境变量已经被定义,并且 nmake 命令所在的目录已经被添加到 MATLAB 的搜索路径中。即保证以下几点:

(1)MsDevDir 环境变量已经定义;

(2)Visual C/C++已正确安装;

(3)nmake 已添加到 MATLAB 的搜索路径中。

而模板 makefile——grt_bc.tmf 是为了使用 Borland C/C++编译器面设计的,使用 grt_bc.tmf 也要保证:

(1)Borland 环境变量已正确定义;

(2)Borland 编译器被正确安装。

7. 普通实时模块

源模块由 makefile——model.m 来自动链接。这个例子中用到的模块如图 9-5 所示,它反映了用于建立普通实时程序的代码模块。

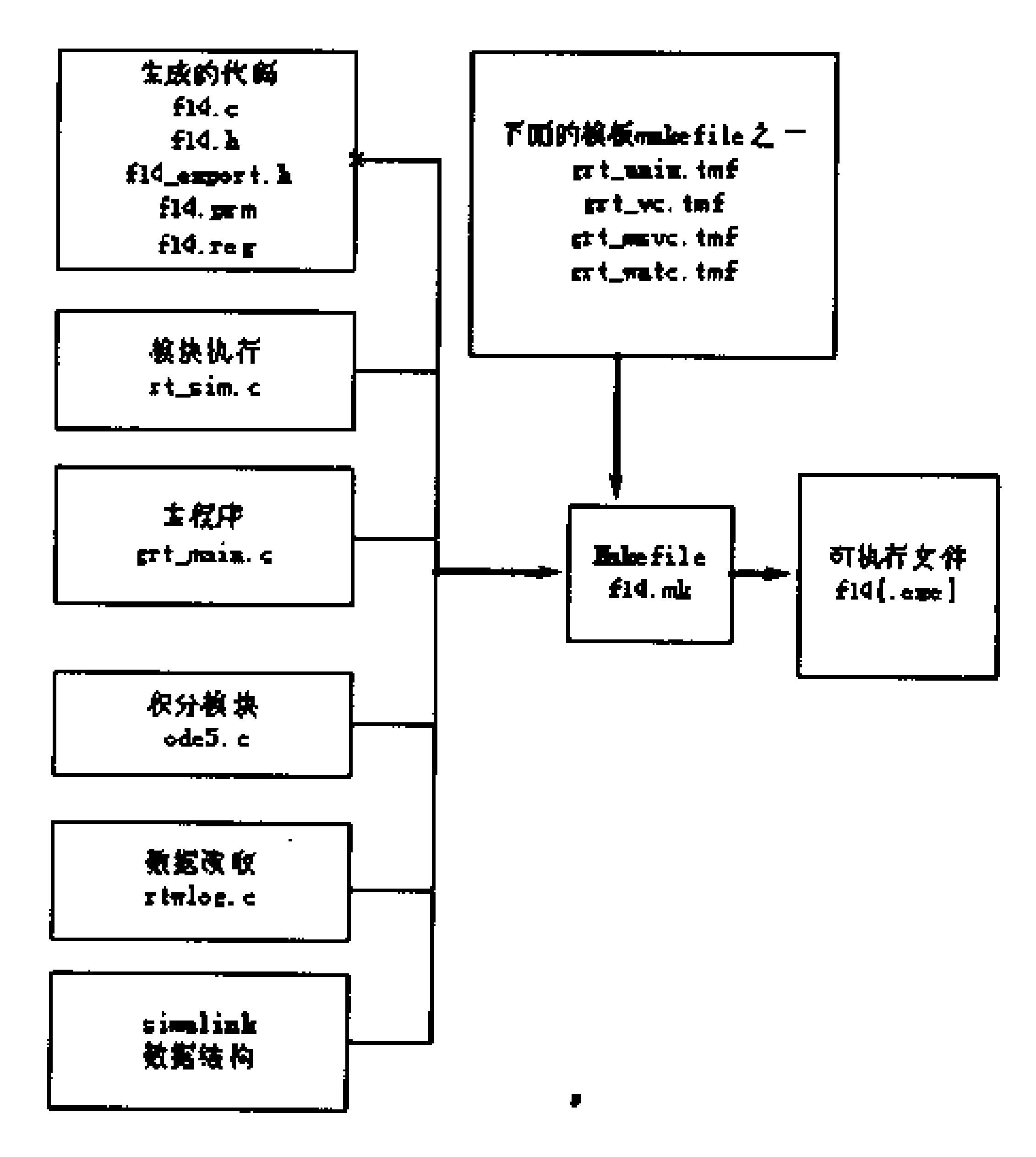


图 9-5　用于建立程序的源代码块

8. 依赖于绝对时间的模块

某些 Simulink 模块使用绝对时间值(即从仿真开始到仿真的当前时刻的时间,注意

这是指仿真模型里的时间表示，而不是运行仿真所用的时间）来计算输出。如果用户正在设计一个有可能会一直运行的程序，那么用户不能使用对绝对实际有依赖性的模块。原因很简单，在时间的指针到了 double 精度数字所能表示的最大值时，问题就出现了。这时，时间就不再增加，于是模块的输出就不再正确了。

表 9－1、表 9－2 和表 9－3 分别列出了所有依赖绝对时间的模块。

表 9－1 依赖绝对时间的模块之一

连续模块	离散模块	非线性模块
Derivative Variable	Discrete－Time Integrator （在触发子系统）	Rate Limiter
Transport Delay		
Transport Delay		

表 9－2 依赖绝对时间的模块之二

接收器	
To Workspace（仅仅在保存的格式是带时间的结构时）	
Scope	
To File	

表 9－3 依赖绝对时间的面模块之三

源模块	
Chip Signal Generator	Discrete Pulse Generator Signal Generator
Clock Ramp	From Workspace Sine Wave
Digital Clock Repeating Sequence	From File Step Code Validation

此外，在仿真参数对话框的 Workspace I/O 页选中 time 这个检查框来保存时间，也需要绝对时间。

9.2.3 代码验证

在完成了可执行程序的建立之后，f14 模型的单机仿真程序版本现在可以和 Simulink 模型进行性能比较了。前面讲过，可执行程序将数据 pilot G forces、aircraft angle of attack 和仿真时间，保存到 MAT 文件。读者可以通过设置 Scope 的 Save data to workspace 选项，把控制输入数据保存到 MATLAB 工作空间。这样，读者就可以用 MATLAB 命令来绘出 MAT 文件所保存数据的曲线图。在 f14 的 Simulink 版本和普通实时单机版本，模拟控制输入用的方波频率为 0.5（rad/s），幅度为正负 1，均值为 0。

请读者先运行 Simulink 仿真模型，仿真的时间从 T＝0 到 T＝60。图 9－6 显示了 Simulink 模型仿真运行输出的结果。然后再运行 RTW 所建立的可执行程序，比较它产生的结果和 Simulink 模型输出的有什么不同。

1. 运行单击程序

下面从 MATLAB 命令窗口运行单机程序。

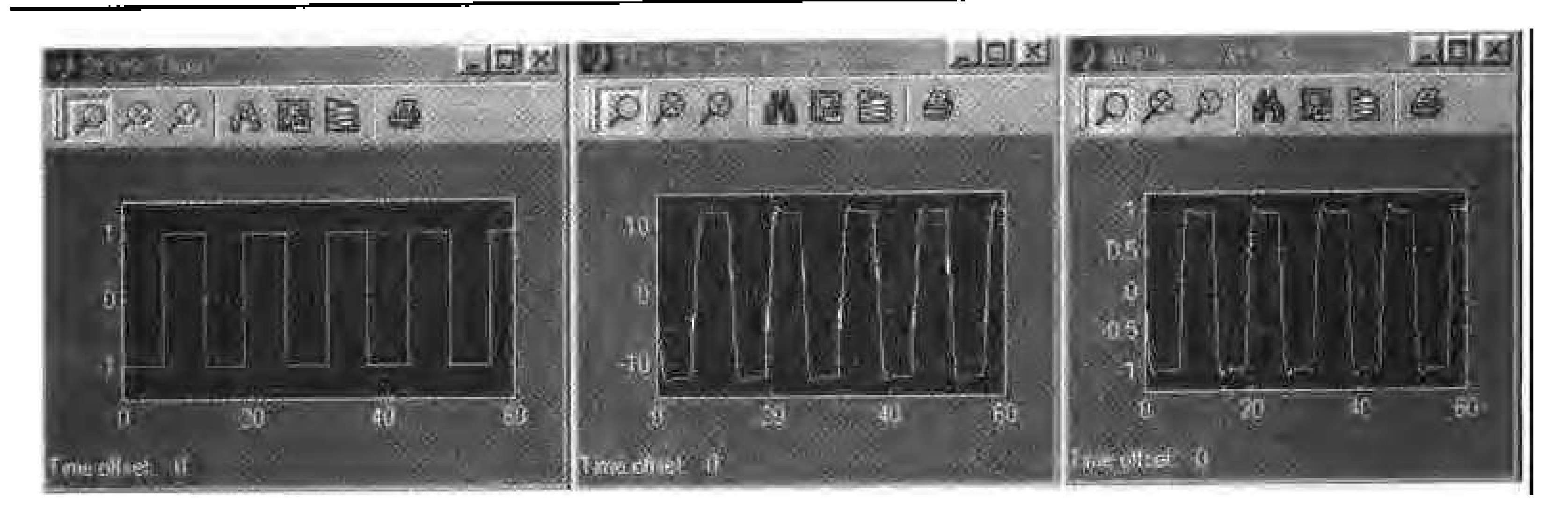


图 9-6 Simulink 模型得到的结果

```
>>! f14
```

在这里的作用是把跟在后面的命令传给操作系统，于是这个命令就是运行 f14 的单机程序版本(而不是 M 文件)。要获得单机程序的运行结果，可以载入 f14.mat 文件到工作空间。

```
>>clear    %先把 Simulink 模型运行后存储到 MATLAB 工作空间的变量消除
>>load f14
```

再查看工作空间中的变量：

```
>>who

Your variables are:
rt_Angle_of_Attack
rt_Pilot_G_force
rt_Stick_Input
rt_tout
rt_yout
```

变量 rt_tout 和 rt_yout 的产生是因为在 Workspace I/O 选中它们相应的检查框的原故，而变量 rt_Pilot_G_force、rt_Angle_of_Attack 和 rt_Stick_Input 这是使用模型 f14 里的 Scope 模块的属性页来保存的。

对使用 Scope 模块不熟悉的读者，可以查阅本章前面的内容。这些变量的命名取决于产生它们的 Simulink 模块。对所有的名称都会加上前缀 rt，表示实时数据。

于是就可以使用 MATALB 命令来绘出三个工作空间变量相对于时间的曲线。

```
>>plot (rt_tout, rt_Stick_Input (:, 2));
>>figure;
>>plot (rt_tout, rt_Pilot_G_force (:, 2));
>>figure;
>>plot (rt_tout, rt_Angle_of_Attack (:, 2));
```

图 9-7,9-8 演示了实时程序得到的仿真结果，其各个变量的排列顺序和图 9-6 相对应。

2. Simulink 模型和普通实时程序结果的比较

读者可能已经注意到 Simulink 仿真和实时代码产生几乎相同的结果。现在就来比

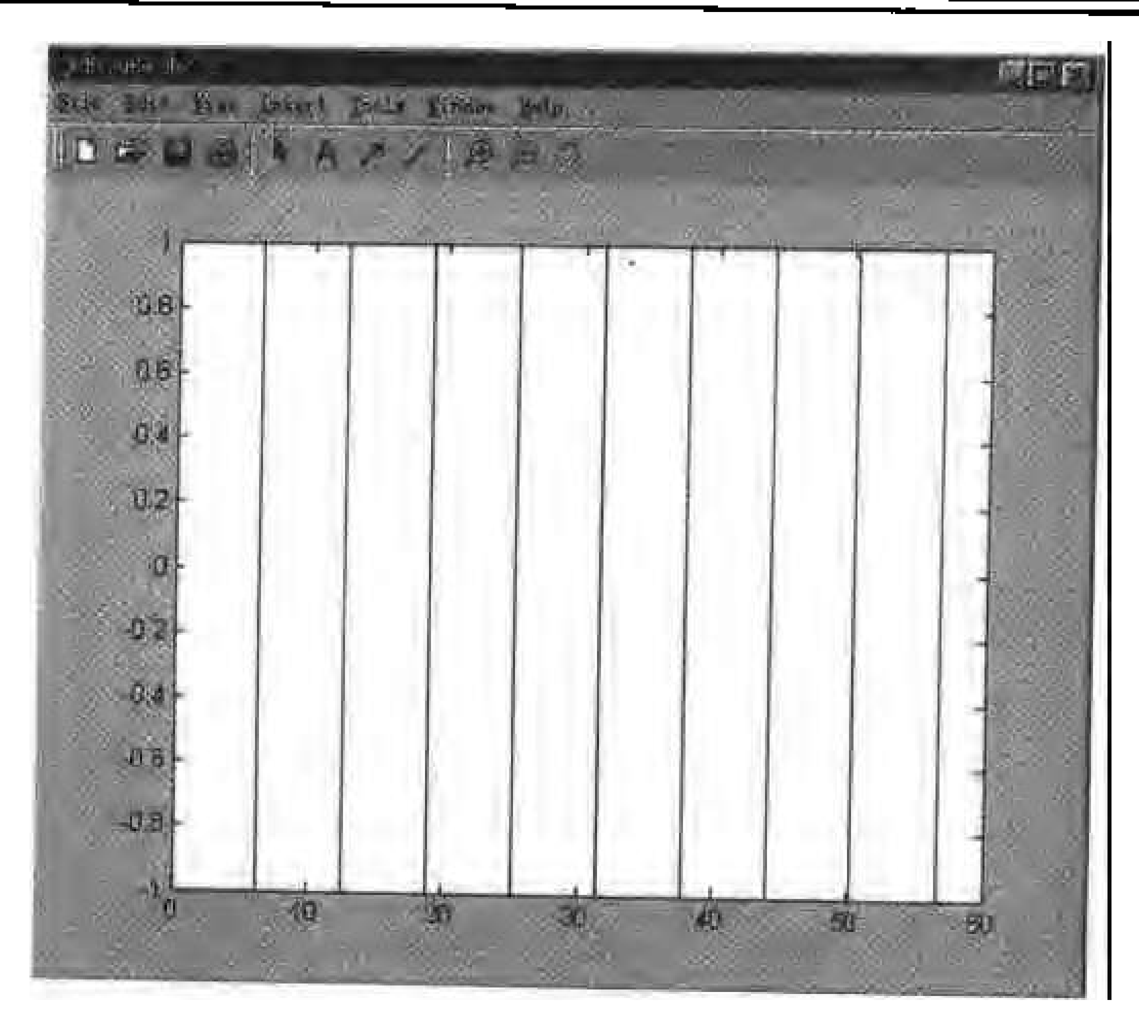


图 9－7　实时程序得到的输入结果

较 Simulink 模型的输出和 Real－Time Workshop 获得的仿真结果。请按下面的步骤进行一个正确的比较：

(1)首先，确信 Simulink 和 Real－Time Workshop 建立过程采用相同的积分机制（必须是固定步长），例如都是 ode5（Dormand－Prince），同时也要把固定步长设置为相同的数值（例如，0.05）。

(2)设置 Scope 模块的 Save data to workspace 选项，以保存 f14 模型的输入和输出（例如，可以让攻击的飞行角度 Scope 模块输出变量为 Angle－of_attack，飞行源所承受重力 Scope 模块输出为 Pilot－G_Froce，控制输入 Scope 模块输出为 Stick－input)。

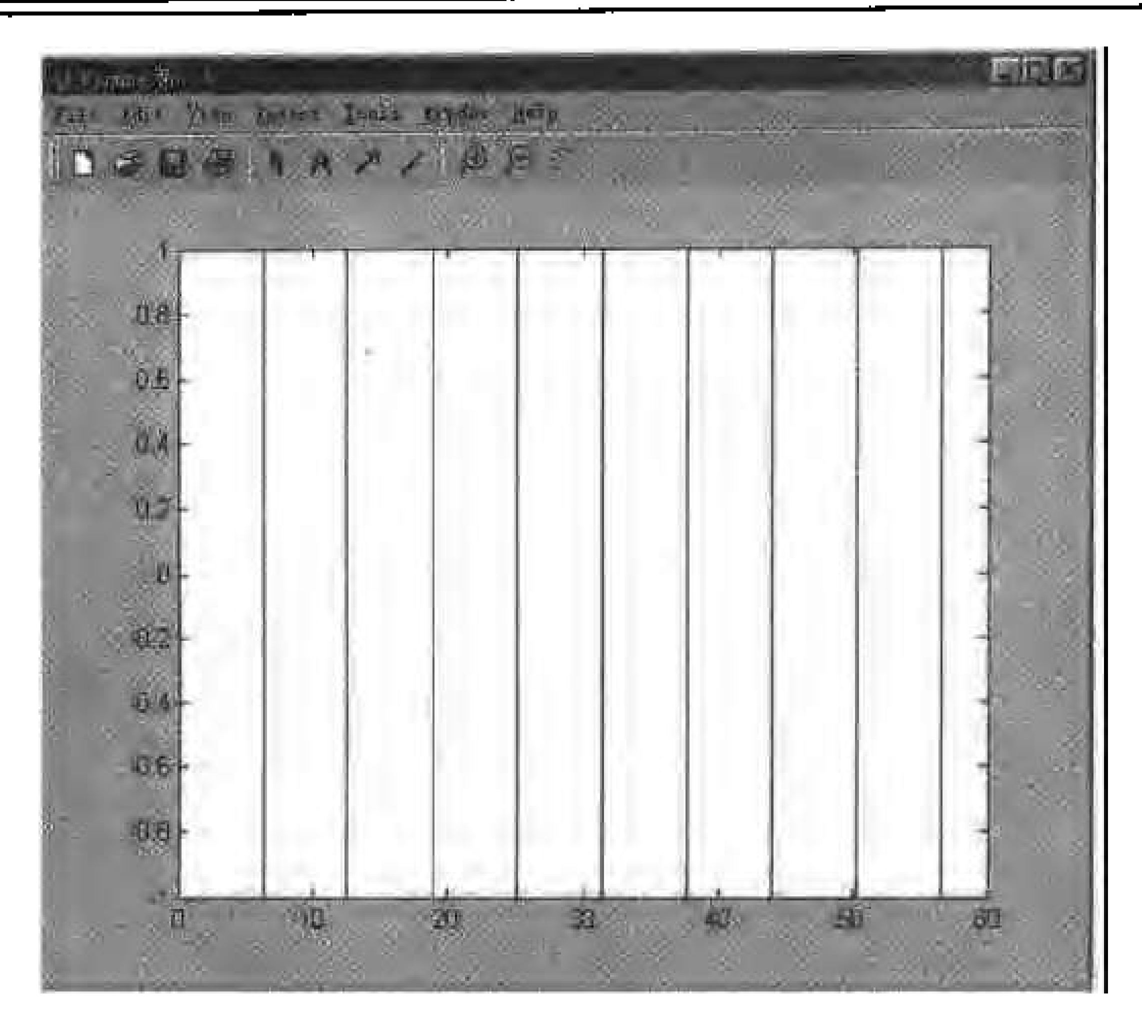
(3)运行 f14 仿真，指用 Simulink 模型。

(4)建立 f14 普通实时系统，运行它，并且载入 MAT－file——f14.dat，到 MATLAB 工作空间。现在，在 MATLAB 工作空间里，就有两种数据，一种由 Simulink 模型产生，而另一种则由 Real－Time Workshop 建立的可执行程序产生。这一点可以使用 who 命令来证实。

```
>> who

Your variables are :
Angle_of_attack rt_Stick_input
Pilot_G_force Stick_input
Rt_Angle_of_attack
Rt_Pilot_G_force
```

比较 Angle_of_attack 和 rtv_Angle_of_attack，得到：



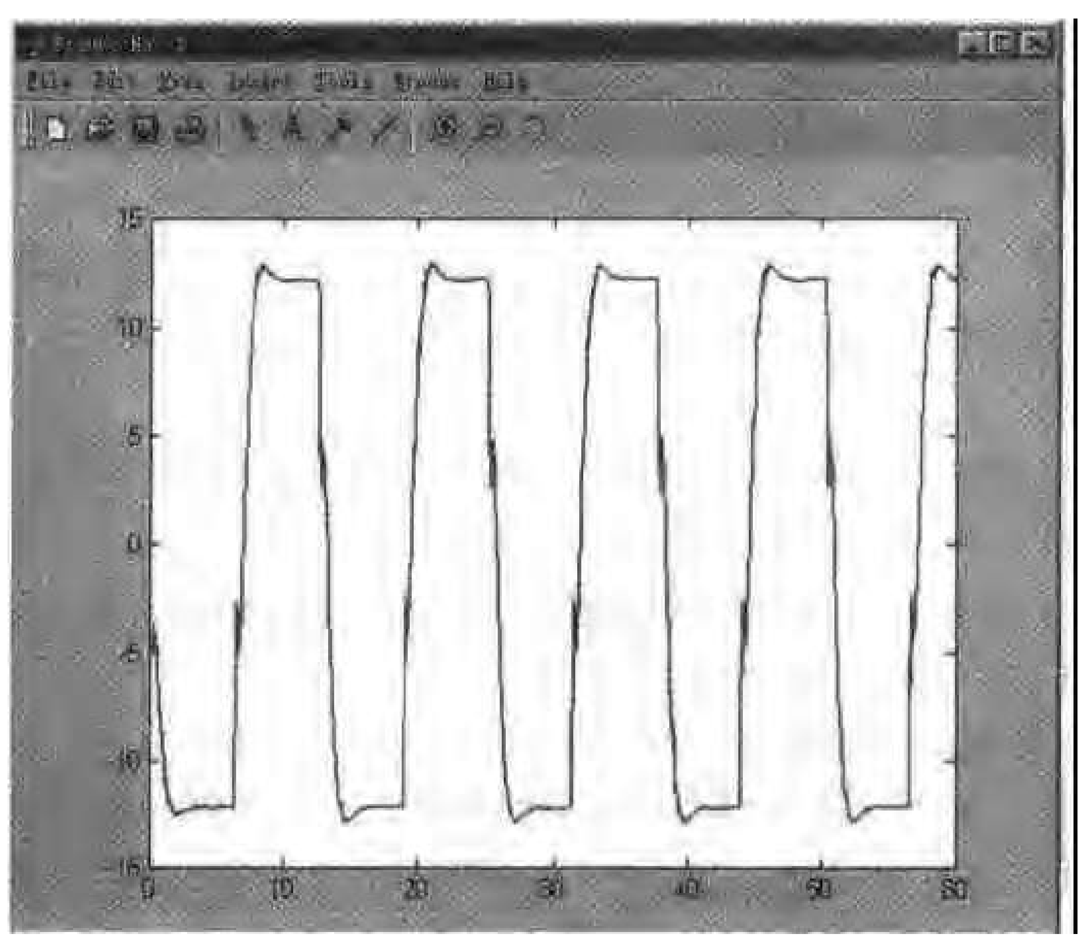


图 9-8 实时程序得到的输出结果

```
>> max (abs (rt_Angle_of_attack - Angle_of_attack ) )

ans =
1.0e-015 *
0          0.5551
```

比较 Pilot_G_force tort Pilot_G_force，得到：

```
>> max (abs (rt_Pilot_G_force - Pilot_G_force) )
```

```
ans =
1.0e-012 *
0         0.1007
```

总的偏差范围在 1～12 以内。这个误差由许多因素导致,包括:

(1)不同的编译器优化;

(2)语句的顺序;

(3)运行库。

例如,计算 sin(2.0)就会因为所用的 C 代码库不同而不同。

如果要进一步的分析数据,用户可以把 To Workspace 模块加入到 Simulink 模型中,于是就可以访问模型中任何模块所产生的数据,并把它们保存到 MATLAB 工作空间变量中去。

9.3 代码生成和建立过程

Real - Time Workshop 的一个作用就是简化了建立应用程序的过程。Real - Time Workshop 的一个特性就是 automatic program building(自动程序建立),它提供了一种标准方法,建立可在不同的目标环境下使用的实时程序。这种方法是单一的,也是受控的,用户可以对它进行自定义。

自动程序建立使用了 make 命令来控制程序的建立,它还使用一个 M 文件从一个可定制的模板来建立一个定制过的 makefile(被 make 命令引用的描述文件)。前面一节介绍了建立的过程和模板 makefile,在这一节将更为详细地讨论这些概念。包括的主题有:

(1)自动程序建立;

(2)Real - Time Workshop 用户界面;

(3)配置生成的代码;

(4)定制模板 makefile;

(5)普通的实时模板 makefiles。

9.3.1 自动程序建立

Real - Time Workshop 自动完成从 Simulink 模型建立一个单机程序的任务。前面讲过,当 Build 按钮一旦被按下,make 命令被调用,从 make _ rtw 开始。调用的过程包括三个主要步骤,它们受一个 M 文件 make _ rtw.m 控制。

(1)生成模型代码;

(2)生成定制一个指定建立过程的 makefile;

(3)调用具有定制过程的 makefile 的 make 命令。

图 9-9 反映了建立的过程。当 Build 按钮被单击之后,上面的这些步骤就被自动完成。

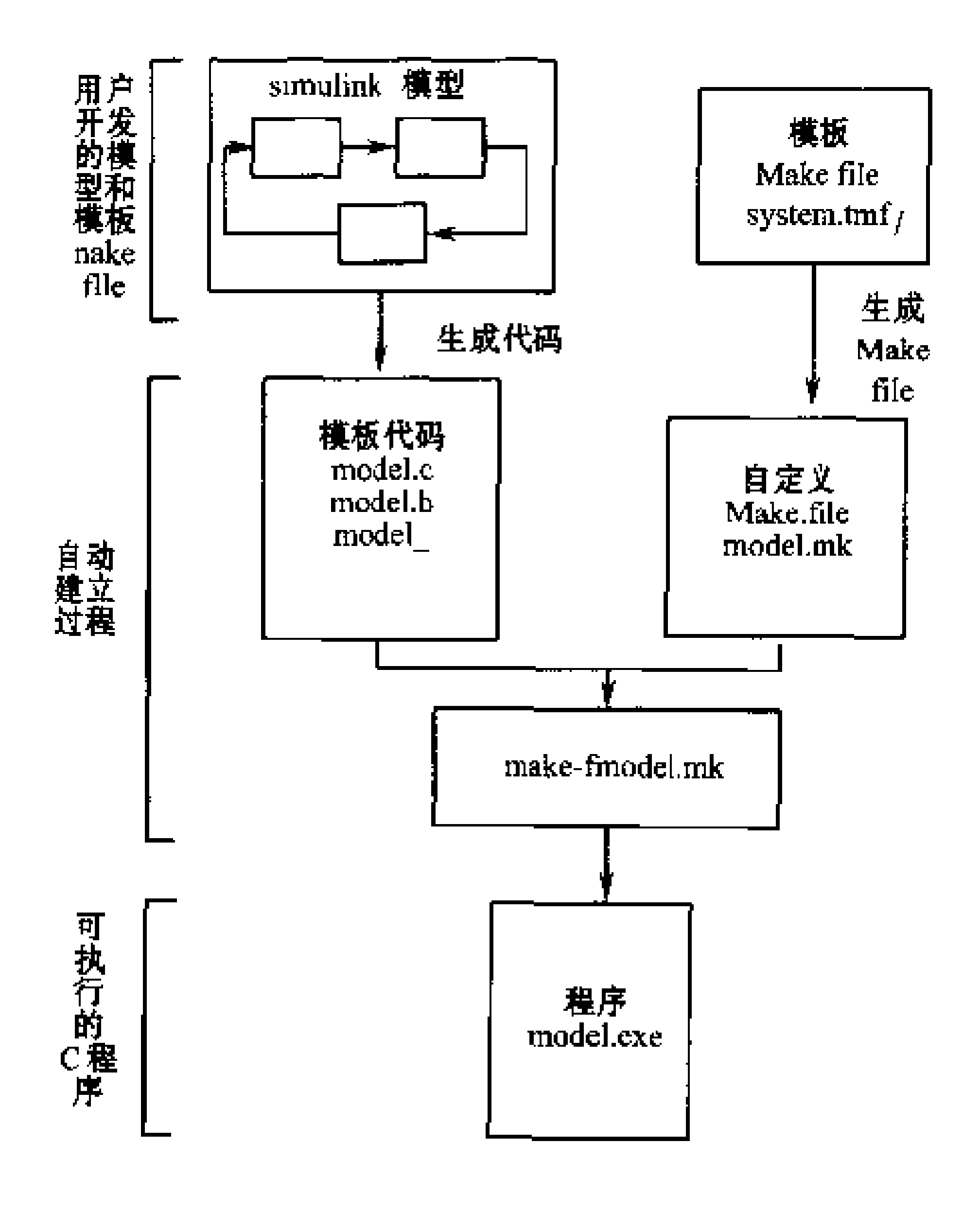


图 9－9 建立过程

9.3.2 Real－Time Workshop 用户界面

为了方便用户使用 RTW，Simulink 提供了一个用户界面来控制 RTW 的执行，它包含在仿真参数对话框里，其中最主要的就是 Real－Time Workshop 页。Real－Time Workshop 页只对 Real－Time Workshop 有效，而其他的页则对 Simulink 仿真和 Real－Time Workshop 都有效。也就是说仿真参数对话框里的所有页面都影响 Real－Time Workshop 实时代码的生成。使用 Real－Time Workshop 时，Solver 页上的解法器(Solver)选项必须设置为固定步长的解法器(fixed－step solver)。Workspace I/O 页是设定数据保存选项的，Diagnostics 页是指定产生错误警告的选项，这些页在本书的有关章节都有详细的介绍。

关于 Real－Time Workshop 页，在前面一节，我们仅仅用到了其中的 Build 命令，其他的编辑框和选项检查框都没有涉及，下面就来具体介绍它们。请读者对照 Real－Time Workshop 页来阅读下面的内容(MATLAB5.3)如图 9－10 所示。

1．系统目标文件

读者可以看到，在该页的代码生成面板(Code generation)，有一个名为系统目标文件(System target file)的编辑框。System target file 用于指定代码的类型和生成代码的目标机器类型。单击编辑框右边的 Browse 按钮可以浏览系统中可以使用的目标文件的列表。该列表还详细给出了各个目标文件的使用场合，例如，缺省的 grt.tlc 适用于普通代码目标。在选定了系统文件之后，用户就可以指定目标语言编译器(TLC)，表9－4列出了常

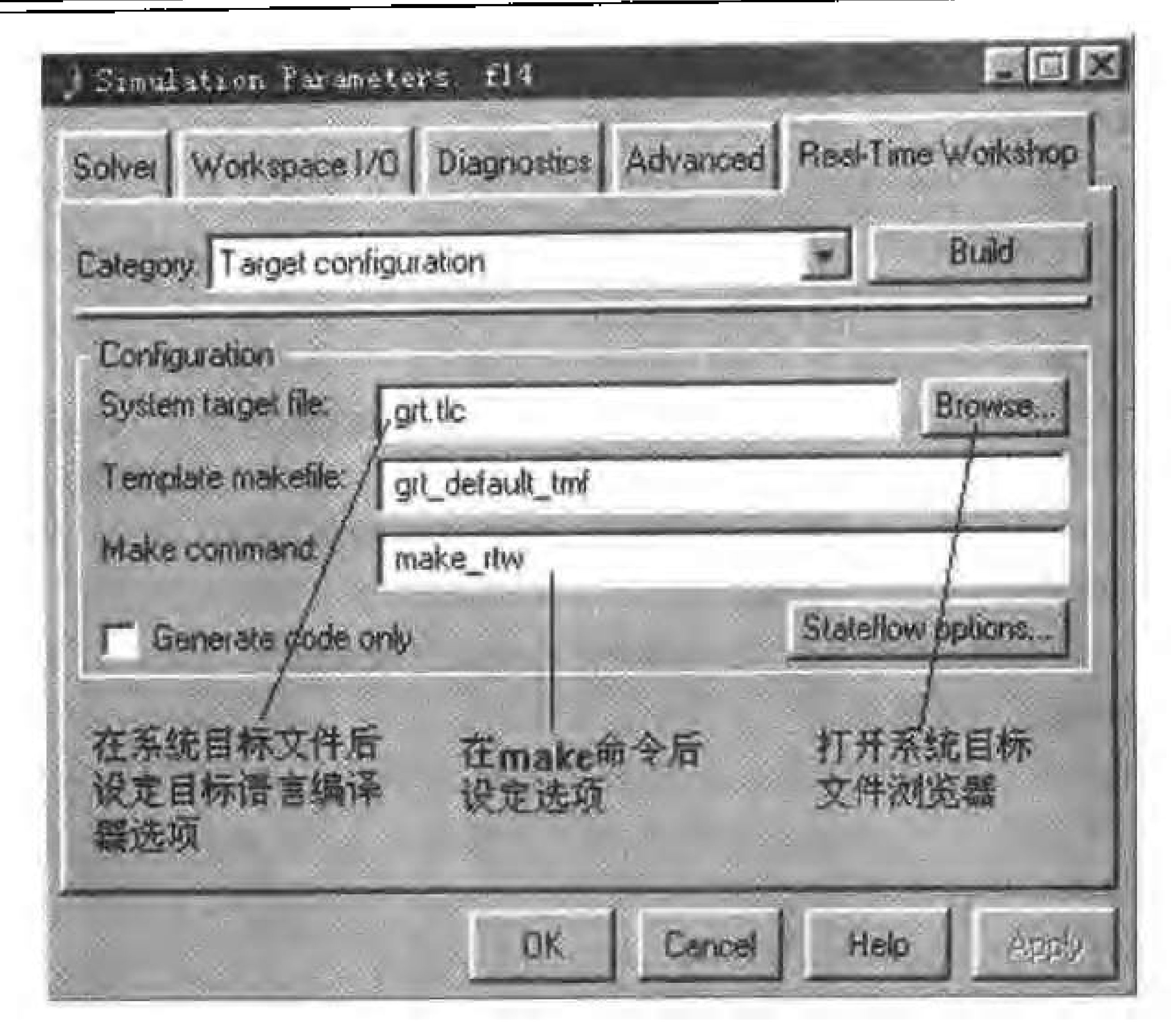


图 9 - 10 Real - Time Workshop 页

表 9 - 4 目标语言编译器的常用属性

选 项	说 明
- Ipath	添加目录到搜索目标文件(后缀名 .tlc)路径列表中去
- m[Nla]	当遇到错误时,报告最大错误(缺省值是 5),例如,- m3 说明了至多 3 个错误被报告,要报告所有的错误,应说明为 - ma。
- d[glnlo]	说明调试模式(generate,normal 和 off),它的缺省值是 off。当 - dg 被指定,一个 .log 文件为用户的每一个 TLC 文件建立。当调试模式被激活,目标语言编译器显示目标文件的每一行遇到的次数
- aVariable = expr	赋给在编译过程中被目标文件使用的变量一个指定的值(例如建立一个参数值对)

用的几个属性。

这些属性的添加方式是在 System target file 域的文件命令后直接添加。更多的选项设置可以在 Code Generation Options 对话框里获得,这个对话框是通过单击 Options 按钮打开的。

2. 内嵌参数

Inlining parameters 指模型中具有常数采样时间的模块在模型执行时被移走。这些模块的输出数据在模型一旦开始就建立。当这个检查框被选中,Simulink 自动地设置所有的内置参数为“不变常数”。一个参数一旦被设置为内嵌参数,就不能在仿真运行过程中动态地变化了,这样可以提高仿真的运行速度。

3. 保持 model.rtw 文件

当用户要修改目标文件时,就需要对照 model.rtw 文件。但是在建立过程完成之后,

model.rtw 文件一般会被删除，选中 Retain.rtw file 检查框可以避免这一点。

4. 模板 makefile

模板 makefile 在 Generate code only 检查框没有选中时使用。模板 makefile 唯一地说明了要生成的可执行文件的目标。用户可以在编辑框里输入自定义的 makefile 文件，但是一般情况下，还是保持缺省值不变。

5. 建立命令

建立选项面板里的 make command 编辑框用来说明编辑建立命令。建立命令通常以 make rtw 开始(除非它被第三方建立命令所替代)。在用户按下 Build 按钮之后，这个命令就被调用。用户还可以传递额外的参量到 make rtw。例如，当用户使用一个 grt * .tmf 文件时，可以改变优化选项。它的设置为：

make _ rtw OPT _ OPTS = "compiler _ specific _ setting"

6. 系统目标文件浏览器

系统目标浏览器在用户按 Browse 按钮后出现，它给出了 RTW 支持的目标文件列表。因为 RTW 支持多种目标和代码格式，为了方便读者，RTW 提供了这个浏览器，让读者可以直接选择合适自己应用的目标。

7. 选项按钮

在仿真参数对话框里有一个选项按钮。按下这个对话框会打开一个代码生成选项对话框，这个对话框会随用户在系统目标文件浏览器选择的系统目标文件变化而变化。图 9-11 显示了当系统目标文件选择为 grt.tlc 时对话框的样子。

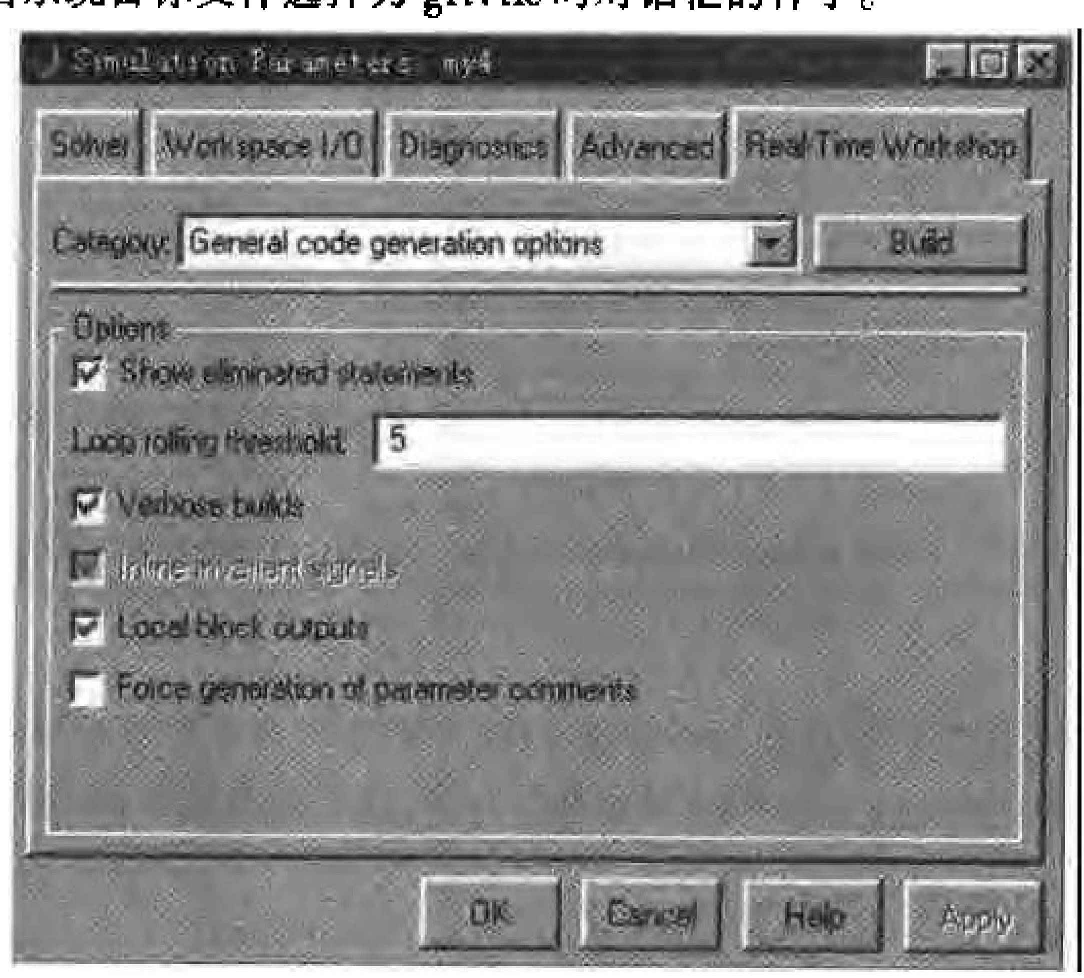


图 9-11 代码生成选项对话框

对话框所支持的特性取决于所选择的目标，但是一般都有下面典型的几种：

(1) Loop rolling threshold;

(2) Show eliminated statements;

(3) Verbose builds;

(4) Inline invarisnt signals;

(5) Local block outputs。

它们的具体作用解释如下：

(1) Loop rolling threshold。让用户设置使循环转换(loop rolling)发生时的门限(一个正整数)。例如，设置这个门限值为 5，意味着如果你的算法被连续调用的次数超过 5 次，那么就把整个算法嵌到一个 for 循环中去。

(2) Show eliminated statements。选中这个检查框，将使得 RTW 把在优化时被删除的语句作为注释语句显示在生成代码里。

(3) Verbose builds。这个特性强迫命令行输出代码生成状况和编译器输出。

(4) Invariant signals。注意"Invariant constants"和"Invariant signals"并不是等同的。"invariant signals"指在 Simulink 仿真过程中保持不变的模块输出信号。例如在图 9 - 12 所示的模型中的 S3 信号就是一个不变信号。如果用户在 Code Generation Options 对话框里选择了 Inline invariant signals，Real - Time Workshop 就把不变信号 S3 内嵌在生成的代码里。这与 Real - Time Workshop 页的 Inline parameters 不同，后者对应应用于不变常数，在这个例子中，如果用户选中了 Inline parameters 检查框，Real - Time Workshop 就内嵌两个常数和增益。

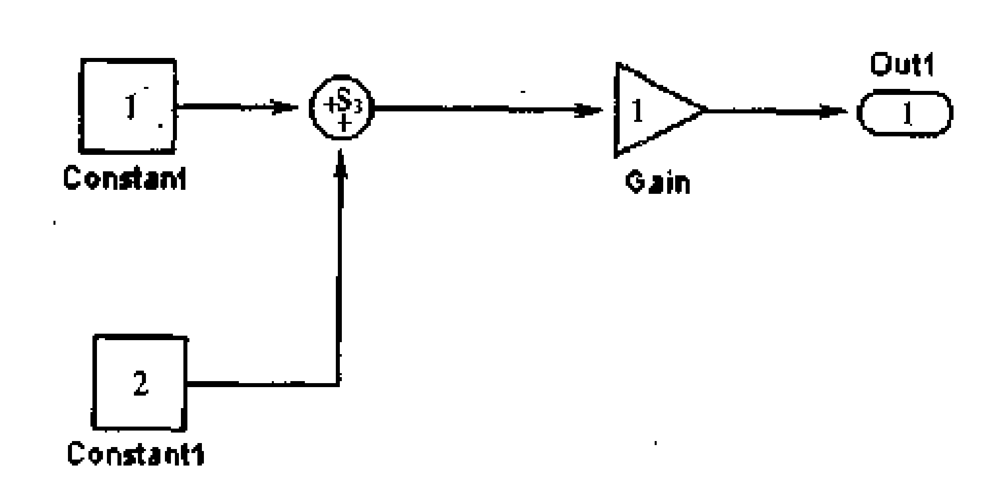


图 9 - 12 不变信号示例模型

注意，对于内置不变信号，用户必须预先选定内置参数，如果 Inline parameters(内嵌参数)检查框没有被选择，Inline invariant 选项就不会工作。然而，也有可能只内嵌参数，而不内嵌不变信号。

(5) Local block outputs。当这个检查框被选中时，Real - Time Workshop 试图放置尽可能多的模块输出信号作为局部变量。

(6) Online Help。把鼠标放在这些字段或者检查框的任何一个时都将激活解释这个特性的用途的提示信息。

在代码生成选项对话框里还可以进行的设置有：MAT - file variable name modifier 选择列表，可以让用户决定是否要往生成的可执行程序运行后所生成的 MAT 文件内的变量名称添加前缀，缺省值为"rt _"，它的另外一个值是"none"。External mode(外部模式)检查框，选中它使生成的实时代码支持外部模式。

通常还会有一个 Function manage 的多项式列表。这个设置的作用是防止所生成代码的某个函数或者某个文件太长，因为某些编译器对单个文件或者函数的长度有限制。

如果在这里选择了 Function split 或者 File split,那么 RTW 将会根据下面的两个编辑框所设置的门限来决定是否将函数(文件)分割。

8. Tunable parameters(可调参数)

当用户在 Advanced 页,选中了 Inline parmeters 检查框,Configure 按钮就变成了可用状态。单击它,就可以打开模型参数配置对话框,如图 9－13 所示。

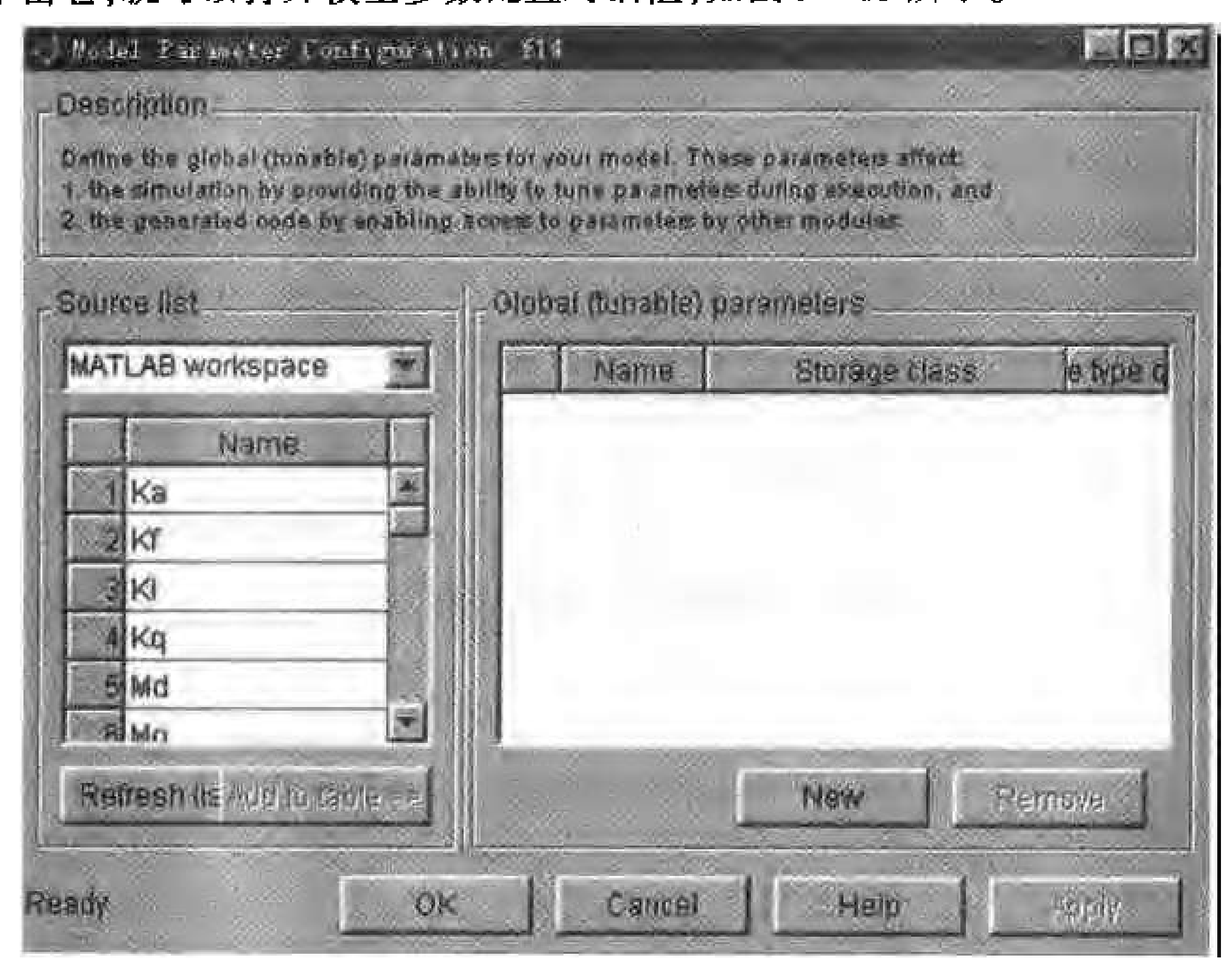


图 9－13　模型参数配置对话框

模型参数配置对话框支持下面的特性：

(1)Parameter tuning——去除 Variable 字段里的任何变量的内嵌特性,可以通过单击 Add 按钮把变量名添加进去。也就是忽略掉所选变量的 Inline parameters 检查框,使之成为可调的,但是其他的参数仍然保持内嵌特性不变。

(2)Storage Class——用户可以改变所选择变量的存储类型。下拉列表里提供的选项有：

• Auto——使 Real－Time Workshop 将变量保存在一个永久数据结构里,这是 RTW 的缺省存储类型选项;

• Exported Global——将变量声明为一个全局变量,这样就可以从生成代码的外部来访问它;

• Imported Extern——将变量声明为外部变量(extern),它必须从生成代码的外面进行声明;

• Imported Extern Pointer——将变量声明为外部指针(extern pointer),它必须从生成代码的外面进行声明。

这些选项在读者想把 RTW 生成的代码和其他的 C 代码(不是 RTW 生成的)连接起来时,就非常有用。

(3)Storage Type Qualifier——这个编辑框输入时,如果对变量进行声明,要预先考虑任何字符串(例如,const)。注意 RTW 不会对字符串进行错误校验。

9. 信号属性

Realt - Time Workshop 对信号支持和参数相同的存储类型选项。改变一个信号的存储类型,可以在模型里选中该信号后用 Edit 菜单下的 Signal Properties 命令来打开信号属性对话框。关于信号属性对话框,本书前面曾讲过一些。这里只介绍和 Real - Time Workshop 有关的选项,它们的位置在对话框的低端。

在对话框里,可以设置的特性有:

(1)Displayable(可显示测试点)——选中这个检查框将使 Real - Time Workshop 为信号分配一个单独点全局存储空间。这可以使测试时减少重写信号数据的可能性。注意一旦选择了这个选项会把 RTW storage class 强置为零。

(2)RTW storage class——可以改变所选中信号的存储类。右边下拉菜单可供选择的选项有:

• Auto——使 Real - Time Workshop 把信号存储为任何它适合的存储类。这是 RTW 的缺省存储类选项;

• Exported Global——把信号声明为可以从生成的代码外访问的全局变量;

• Importec Extern——把信号声明为外部变量(extern),于是它必须从生成代码外部来声明;

• Imported Extern Pointer——把信号声明为外部指针,于是它同样要在外部被声明。

这些情况用在用户想把 RTW 代码和其他的 C 代码(即不是用 RTW 生成的代码)嵌套使用时,用户注意要负责把代码块正确地链接在一起。

(3)C API for Signal Monitoring. RTW——提供了开发独立于外部模式的信号监测的 C API See。

9.4 外部模式

9.4.1 介绍

外部模式是 RTW 提供的一种支持在实时环境下动态改变参数的仿真模式。"External"是指这种模式涉及两种独立的计算环境、一个宿主和一个目标,外部模式允许这两个独立的环境相互通信。宿主是 MATLAB、Simulink 和 RTW 运行的计算机,而目标则是 RTW 建立的可执行程序运行的计算机。使用外部模式,用户就可以改变模块参数并且在 Scope 模块查看和保存模块输出。外部模式实际上就是在 Simulink 和 RTW 生成的代码间提供一个通信通道。这个通信通道可以是网络协议,例如 TCP,或者是共享内存。

在外部模式中,宿主(Simulink)等待参数或者信号(来自外部)的变化,一旦接收到这种变化,它就发送请求目标接收参数变化的消息或者是上载信号数据,而目标响应这种请求之后,Simulink 再把新的参数值提供给目标机器。由此可见,外部模式通信是基于客户机/服务器体系结构的,其中 Simulink 是客户机,目标是服务器。

外部模式的主要功能有：

(1)实时地修改或者调整模块参数。在外部模式，无论用户什么时候在模块图表里改变参数，Simulink会自动地把它们下载到正在执行的目标程序中去。这样就允许用户不需要重新编译就可以调整程序的参数。在外部模式，Simulink变成了目标程序的图形前端。

(2)使用多种模块和子系统，查看和记录模块输出。用户不需写特殊的接口代码，就可以监视或者记录从正在执行的目标程序输出的信号数据。外部模式允许用户定义数据从目标上载到宿主的条件。例如，数据上载可以由一个正向的过零点信号来触发，类似地，它也可以由用户手动触发。

外部模式的工作机制是在Simulink模型和RTW生成代码之间建立一个通信通道。这个通道由一个处理消息的物理传送的低层传输层来实现，而Simulink和生成代码都是和这个层独立的。这种设计使得不同的目标使用不同的传输层成为可能。例如，GRT和Tornado目标使用TCP/IP来实现宿主和目标间的通信；而xPC目标支持RS-232(串口)和TCP/IP通信；Real-Time Widows Target则是通过共享内存来实现外部模式(Real-Time Windows Target是和RTW独立的产品)。

这一节将介绍：

(1)如何在自己的机器上建立外部模式。这时，宿主和目标在同一台机器上；

(2)如何使用包含在Simulink里的基于TCP socket的外部模式实现；

(3)参数调整；

(4)使用Simulink Scope模块进行信号查看和保存。

在学习本节之前，先请学习一下前一节“代码生成和建立过程”。

9.4.2 使用grt(普通实时目标)的外部模式入门

这一小节将通过一个使用普通实时目标(grt)示例，一步一步地向你介绍外部模式的入门知识。这个例子的宿主和目标处在相同的机器，它不需要读者准备另外的机器来运行可执行程序，就可以体验一下什么是外部模式。如果读者的兴趣仅此为止，那么这一节后面的内容，就可以先放在一边。

在进入外部模式的建立之前，请读者先建立一个简单的模型来作为生成代码的模型。读者可以使用本书的示例，也可以自己建立一个模型，这不影响这里的介绍，图9-14显示了本书采用的简单模型。

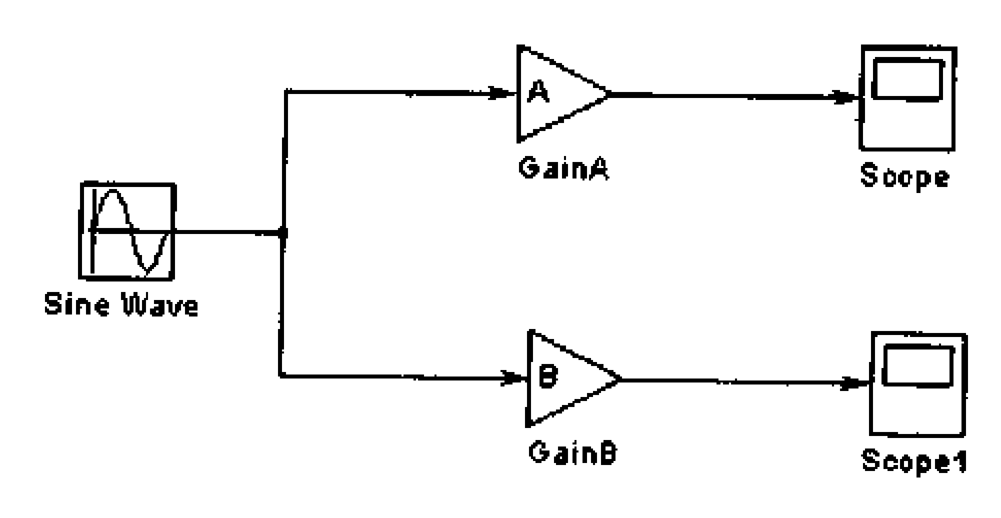


图9-14 外部模式的示例模型

下面就来为这个模型建立外部模式目标程序，其步骤为：

(1)使用 Simulation 菜单下的 Extenal 命令把仿真设为外部的。

(2)打开仿真参数对话框。在 Solver 页，把 Solver Options Type 置为 Fixed - step(固定步长)，并且把 Fixed - step size(固定步长大小)置为 0.01，而仿真时间 Stop time 依旧是缺省保持不变，最后置 Decimation 为 1，如图 9 - 15 所示。

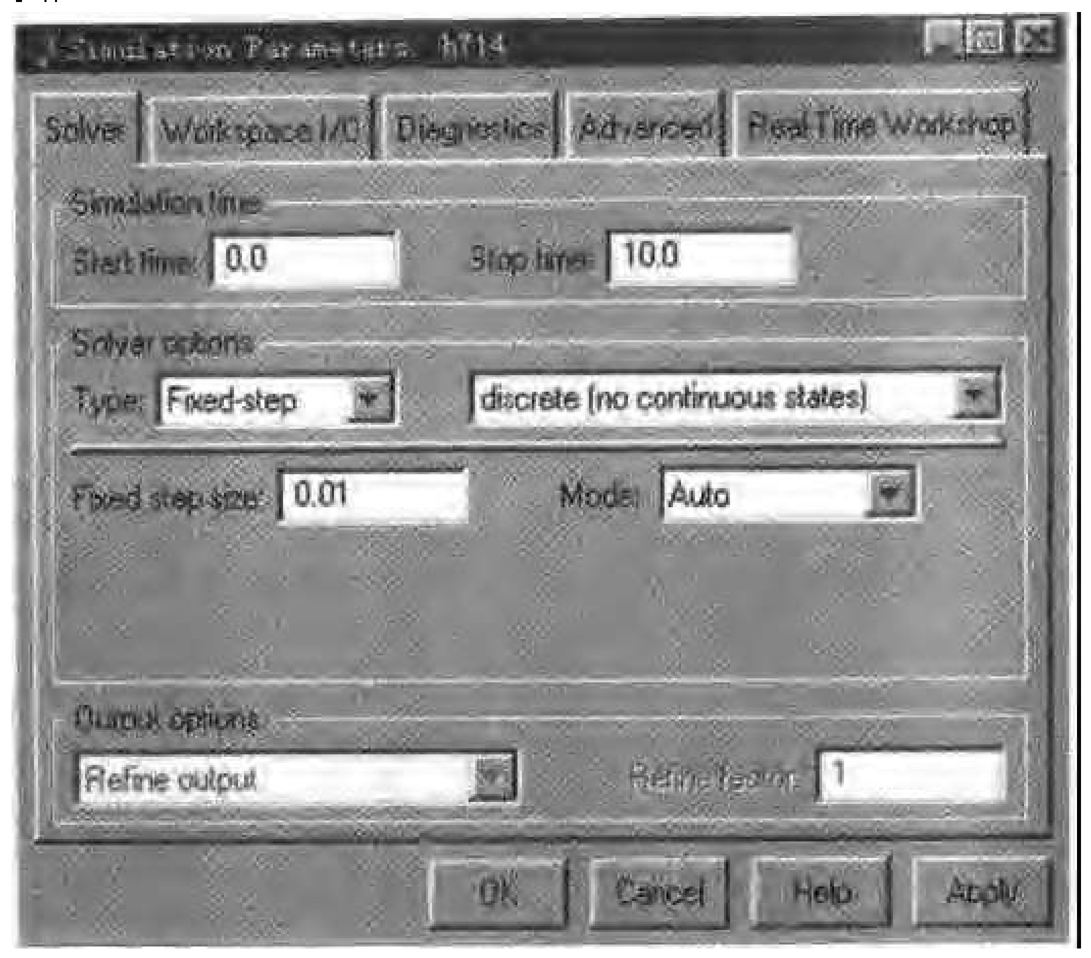


图 9 - 15　Solver 页的设置

(3)在 Workspace I/O 页，去掉 Time 和 Output 检查框的选中状态，因为在这个例子中，数据不需要保存到工作空间或者 MAT 文件中，如图 9 - 16 所示。

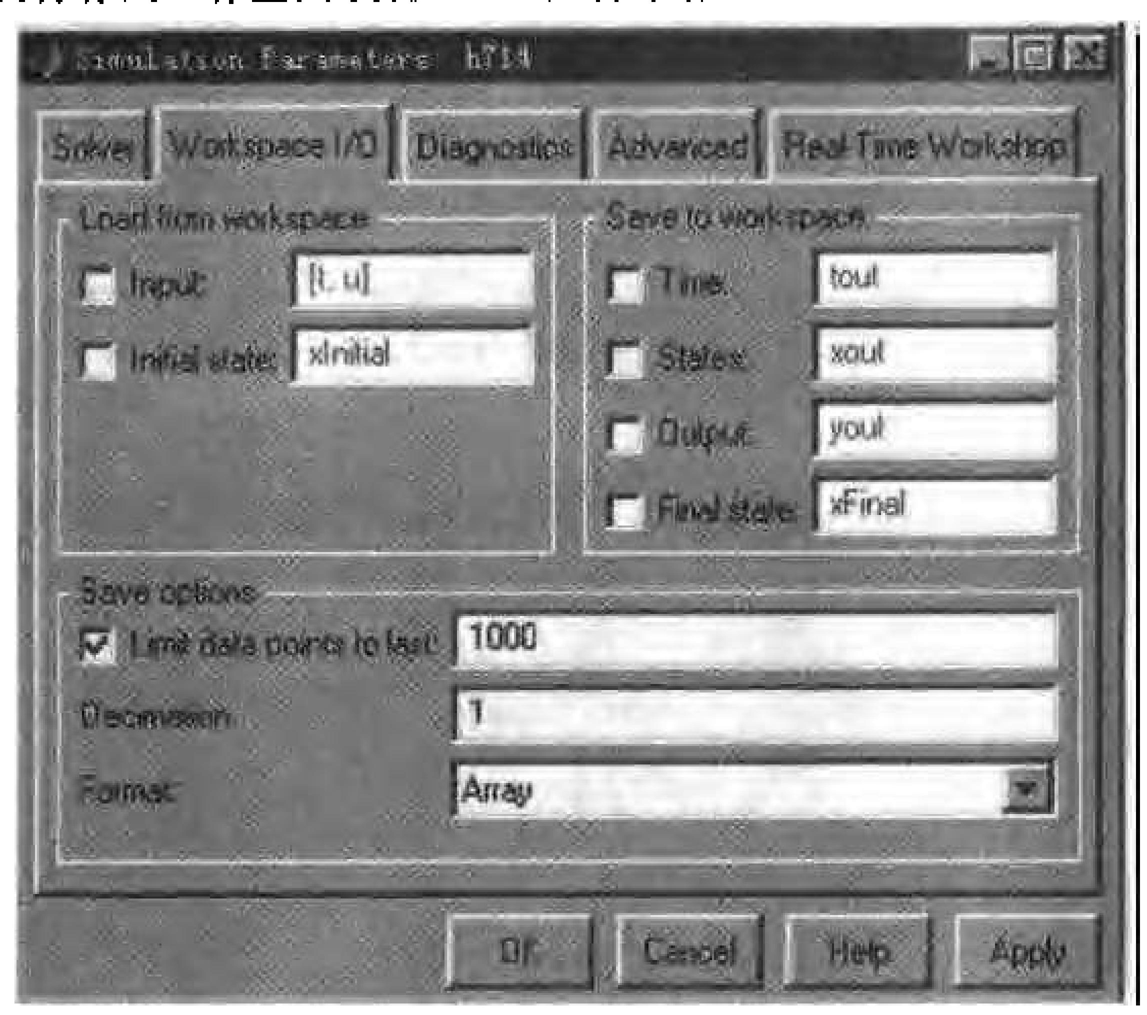


图 9 - 16　Workspace I/O 页设置

(4)在 Real – Time Workshop 页,从 Category 菜单选择 Target configuration 选项。而至于系统目标文件(system target file),缺省情况下,应该指定为普通实时目标(即 grt.tlc),所以就不用修改。如果没有设定为 grt,读者可以从 Browse 打开的系统目标文件列表里选择它。最后完成的 Realt – Time Workshop 页应该如图 9 – 17 所示。

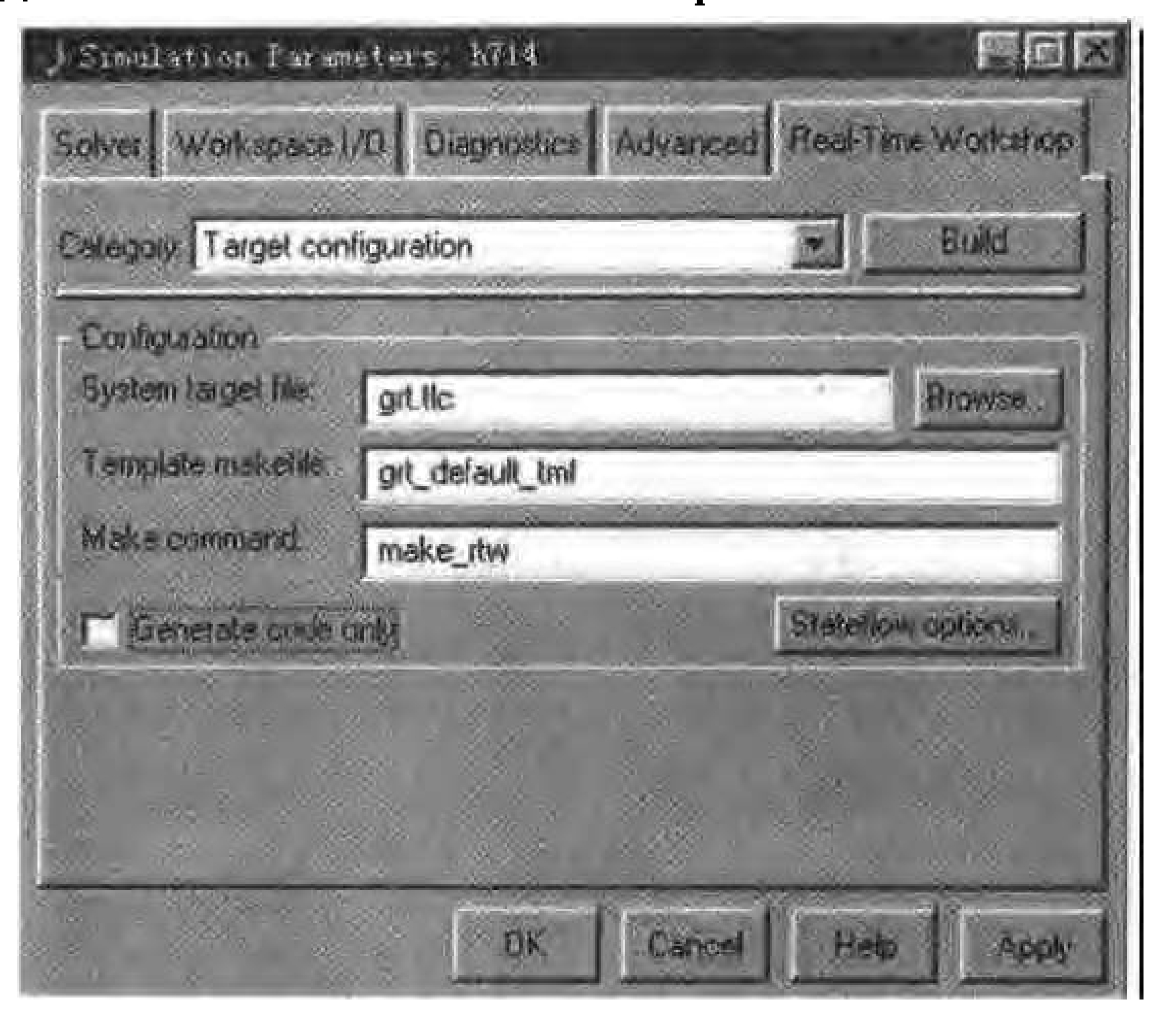


图 9 – 17 Real – Time Workshop 页设置

(5)从 Category 菜单选择 GRT code generation options 选项,并在相应出现的选项面板里选中 External mode 检查框,如图 9 – 18 所示,这样 RTW 就会生成支持外部模式的代码。

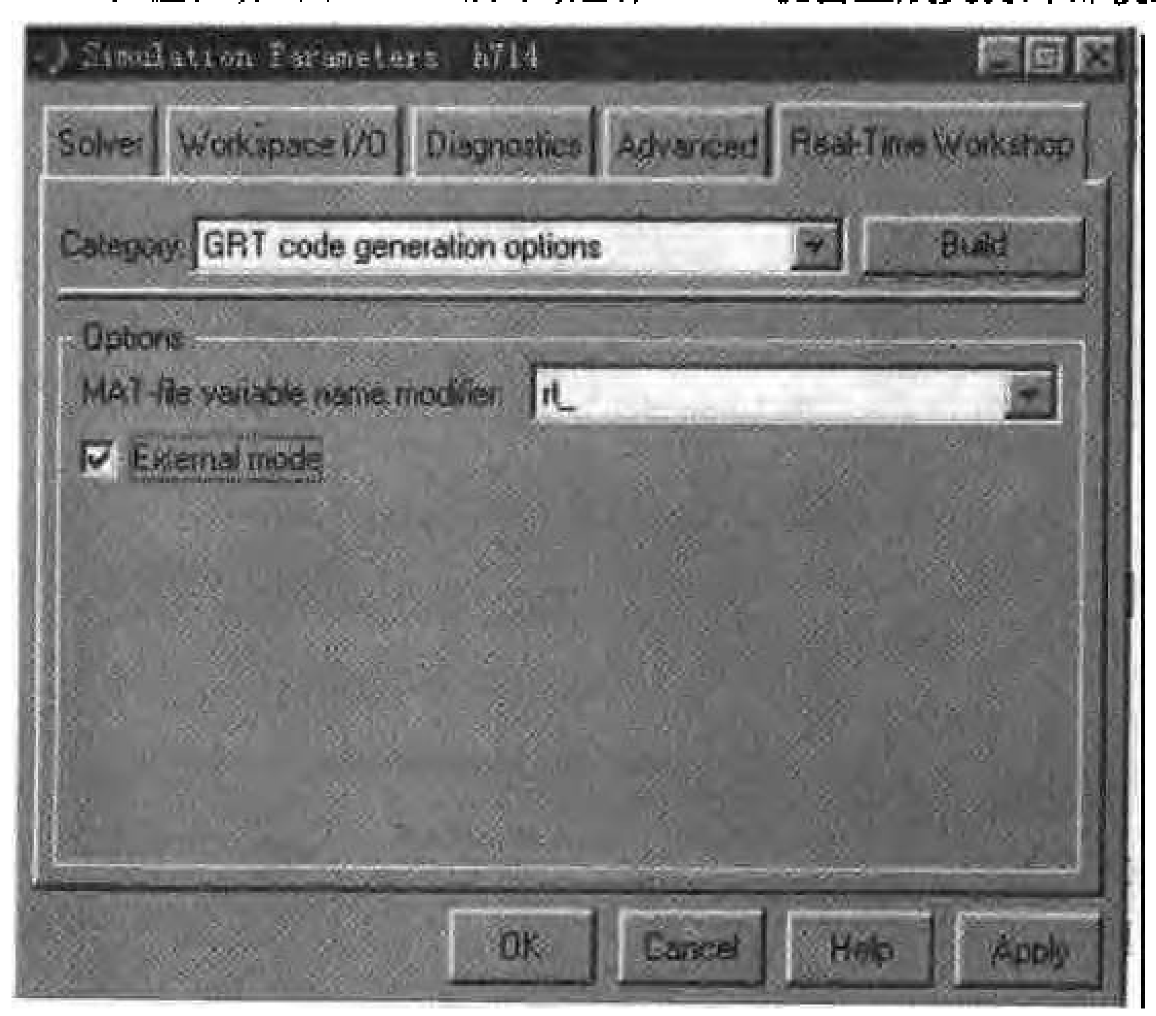


图 9 – 18 设置 GRT 代码生成选项

(6)选择 Advanced 页,确保 Inline parameters 检查框未被选中。外部模式不支持内嵌参数选项。

(7)设置外部模式的特性。用 Tools 菜单下的 External Mode ControlPanel 命令打开模式控制面板,读者可以通过它方便地浏览外部模式要用到的所有特性。外部模式面板能让用户配置宿主和目标之间的通信机制、信号监视以及数据记录等特性,也能让用户将 Simulink 模型连接到目标程序,开始和结束模型代码的执行。图 9 - 19 是外部模式控制

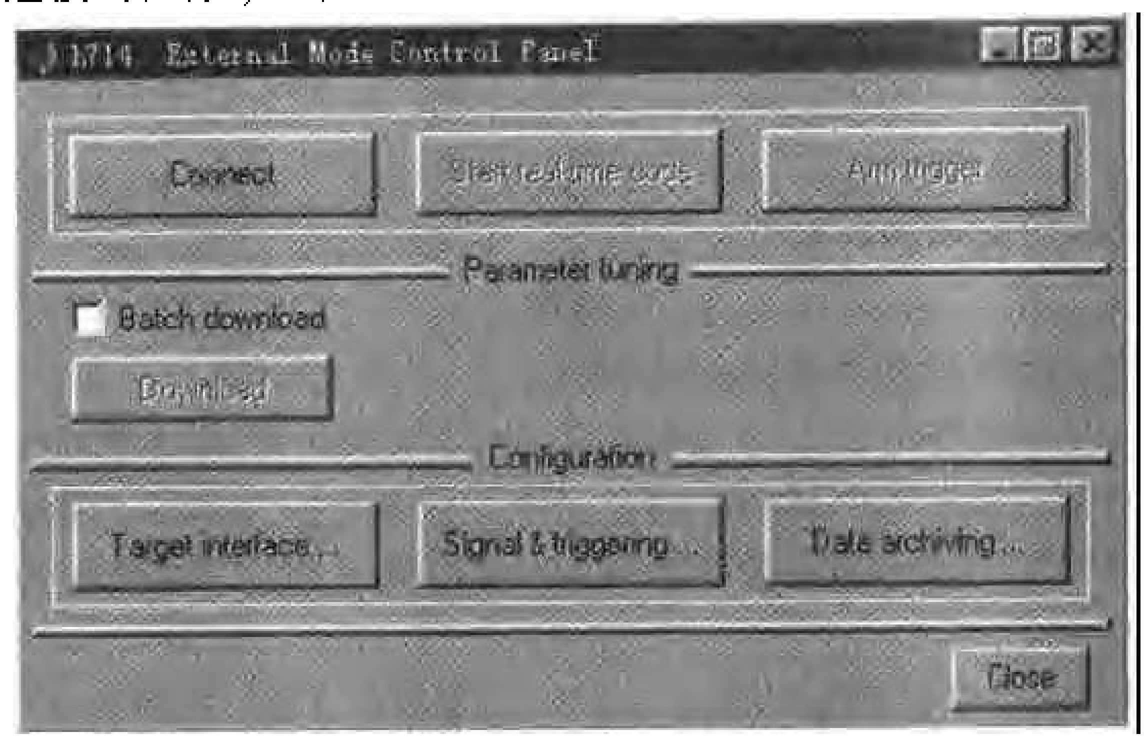


图 9 - 19 外部模式控制面板

面板的样子。图中的上面四个按钮在读者运行了实时程序后使用,而下面的三个按钮将分别打开三个不同的对话框。它们分别是:

• Target interface 按钮打开 External Target Interface 对话框(外部接口对话框),它用来配置外部模式的通信通道。

• Signal&triggering 按钮打开 External Signal&Triggering 对话框(外部信号和触发对话框),它配置哪个信号被查看并且它们如何被触发。用户在开始执行实时程序前必须先在对话框里选择信号。触发一词表示信号何时被捕获和显示。

• Data archiving 按钮打开 Exteranl Data Archiving(外部数据记录)对话框。

(8)按 Target Interface 按钮,打开目标接口对话框,如图 9 - 20 所示。Simulink 要求用户在外部模式下运行模型前,必须先配置目标接口和设置各种信号触发和数据记录选

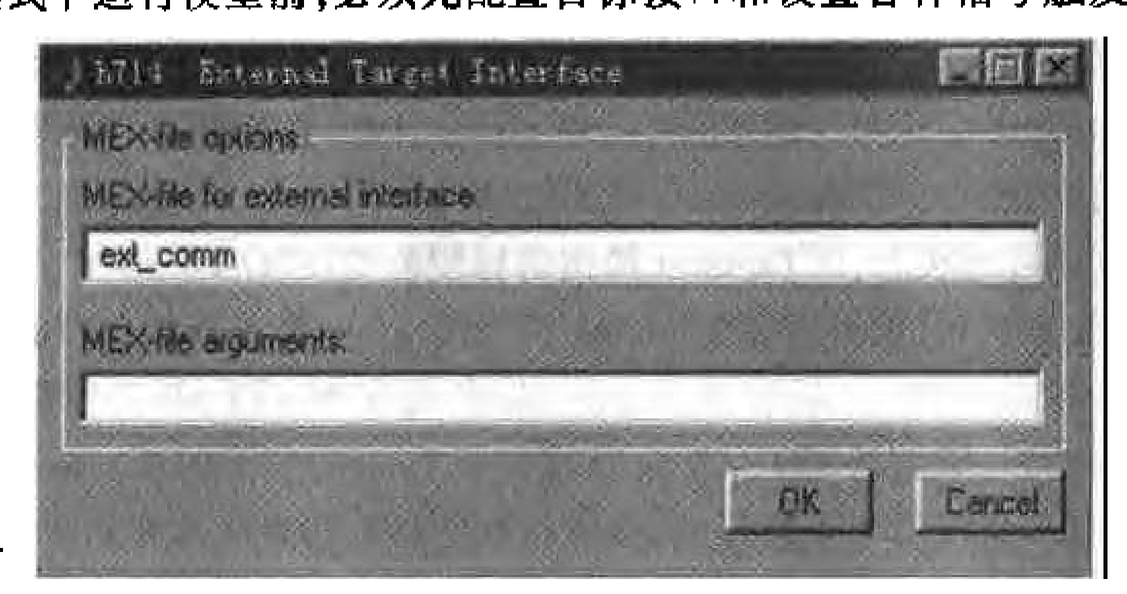


图 9 - 20 目标接口对话框

项。缺省情况下，作为外部接口的 MEX 文件应该设置为 ext _ comm.。这个文件支持使用 TCP 通信协议的宿主和目标之间的通信。MEX 文件的参量可以留作空白。设置完毕后，请按 OK 按钮，退出外部目标接口对话框。

(9)设置信号触发和数据记录。这是启动外部模式的先决条件之一，读者可以单击 Signal & triggering 按钮来打开外部数据保存配置对话框。本例中，读者可以按下面来配置这个对话框：

• 单击选择 Select All，这会选中示例模型中的两个模块，使我们能够实时地查看输出的信号。

• 在触发面板，设置 Trigger source 为 manual(手动)，Trigger Mode 为 normal，持续时间为 1000，以及延迟为 0(没有延迟)。

• 选中 Arm when connect to target 检查框。

(10)关闭外部模式控制面板，并保存模型。

(11)单击 RTW 页上的 Build 按钮在外部模式下生成代码，并建立可执行程序。在按 Build 按钮之前，用户必须先在 MATLAB 工作空间设置增益 A 和 B 的值，这里不妨使用 A =3 和 B = 3(关于模块参数估值请看 Simulink 详解一章)。至此，建立外部模式程序的整个过程就完成了。

在这个例子中，数据记录选项(data archiving)不需要设置。在用 Build 命令生成代码前，请读者先检查是否给 A，B 赋值，以及是否设置了环境变量。而上面的第 9 步操作对可执行程序的建立没有影响，它可以放在建立好的目标可执行程序后进行。

接下来就可以运行建立的目标可执行程序了。请读者在 MS－DOS(Unix 对应的是 Xtem)命令行下输入

```
modelname -tf    inf  -w
```

或者将前面的命令加上“!”，后面加一个“&”，就可以直接在 MATLAB 命令窗口输入。例如

```
! modelname -tf    inf  -w &
```

在这里，模型名称是 exter _ mode。命令中后面的 - tf 开关将重载 Simulink 模型仿真参数对话框中设置的结束时间。把结束时间值设为 tftf，表示模型的运行时间是不确定的，模型会一直运行至接收到一个来自 Simulink 的消息为止。 - w 开关的意思是让目标程序在接收到宿主发出的开始实时代码的消息之前，处于等待状态。如果用户想从程序开始时就观察它的输出，或者是在程序执行前改变参数，这个开关就是必需的。

然后，选择 Simulink 菜单下的 Connect to target 命令，这个命令实现初始化 Simulink 和目标代码之间的握手。当 Simulink 和目标代码连接上之后，菜单下的 Start real - time code 命令就会变成可用状态，而 Conncet 按钮的标题会变成“Disconnect”。选择这个 Start real - time code 命令之后，就可以在 Simulink 模型的 scope 模块查看仿真的结果。

读者可以动态地改变增益模块的增益值，或者是在 MATLAB 工作空间设置新的增益值。在后一种情形，请键入 Ctrl - D 命令或者用 Edit 菜单下的 Update Diagram 命令，在改变 MATLAB 变量值后来更新模型，按这种改变参数的方法称为 downloading。

类似地，读者可以改变正弦波的频率、幅度或者相位。但是读者不能改变正弦波模块的采样时间。因为模块采样时间是模型结构定义的一部分，因为它是生成代码的一部分。所

以,如果读者要改变一个模块的采样时间,那就必须重新编译模型,重新建立可执行程序。

9.4.3 外部模式 GUI

外部模式 GUI 是 RTW 提供的一个扩展图形的用户界面,它是独立的窗口,用来支持外部模式里的特性。这四个窗口分别是:

(1) 外部模式控制面板;

(2) 外部目标接口对话框;

(3) 外部信号和触发对话框;

(4) 外部数据记录对话框。

下面将逐一介绍这些特性,尽管有一些我们在前一节已经提到过,但是在这一节将会给出更详细的内容。

1. 目标接口

按 Target Interface 按钮将激活 External Target 对话框。它用来设置 MEX 文件选项。

(1)作为外部接口的 MEX 文件——外部接口文件的名称。缺省值是 ext_comm,它是为使用 GRT 和 Tornado 目标提供的基于 TCP/IP 的外部接口。而 Real-Time Windows Target 使用的接口文件是 win_tgt,对于客户或者第三方目标,它们可能使用其他的外部 MEX 文件。

(2)MEX 文件参量——外部接口文件的参量。例如,基于 TCP 的外部接口 MEX 文件 ext_comm,包含三个附带参量:目标的网络名、冗长等级和 TCP 服务器端口数目。缺省情况下,这个域留作空白。对于 ext_comm,读者可以在编辑框内输入本地目标名称、长度等级和端口号 17725,即:'localtargetname',1,17725。

2. 外部信号和触发

图 9-21 是外部信号和触发窗口的示意图,这个窗口提供以下的支持:

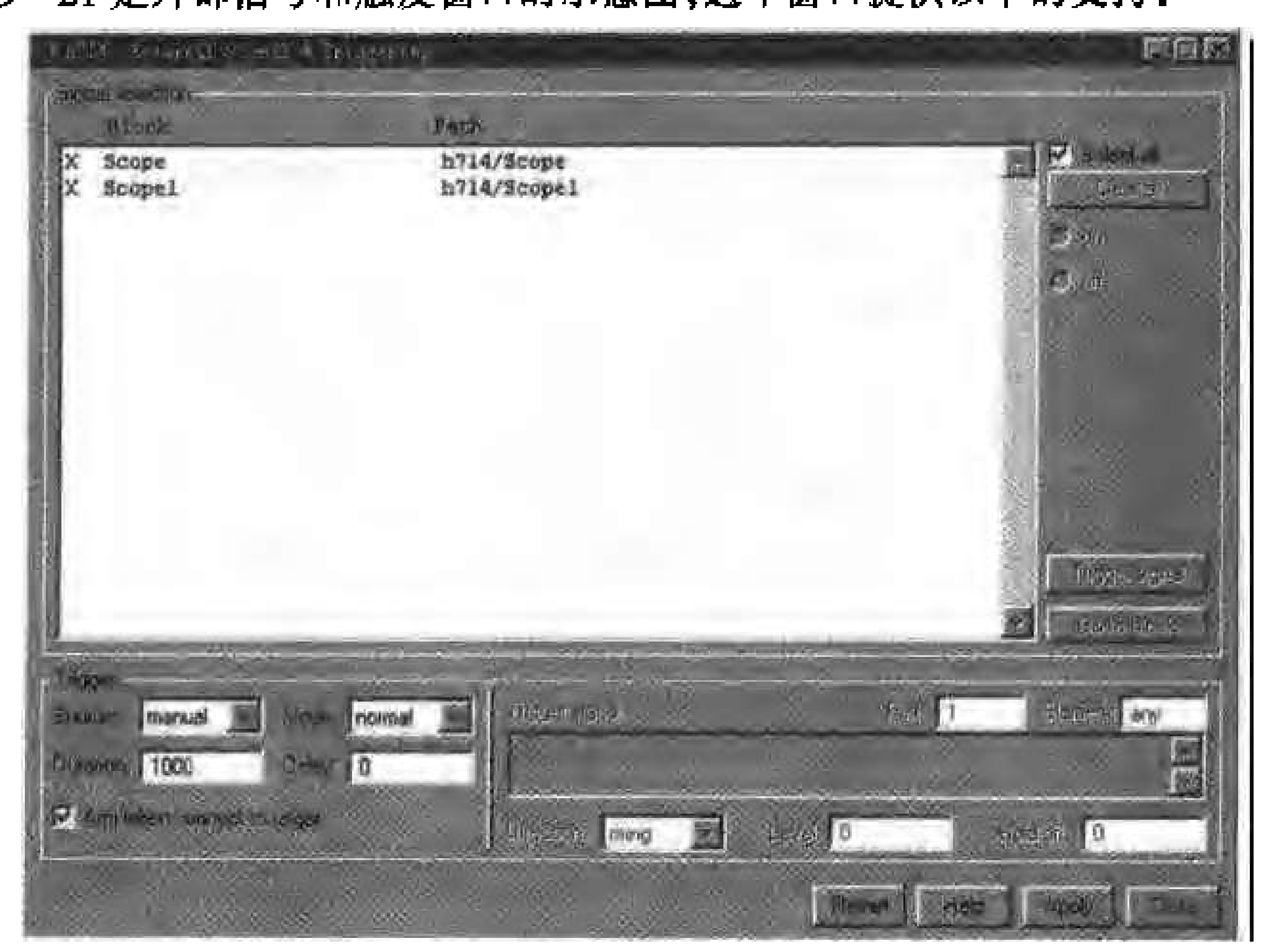


图 9-21 外部信号和触发窗口

(1)信号选择——说明哪个 Scope 模块在外部模式是激活状态。

(2)触发选择——设置这个信号何时在 Scope 模块显示数据。

(3)触发选项——配置如何并且何时在 Scope 模块显示数据。

外部模式的信号选择只限于 Scope 模块的输入信号,用户模型中的任何 Scope 模块都会出现在信号选择列表里。Select all 和 Clear all 按钮分别选择和不选择所有列出的信号。一个信号被选中时,即在模块的名称左边出现一个 X。当一个信号被选择时,如果用鼠标单击信号列表中的模块名称,静态按钮将会被激活。

读者可以通过在信号列表中选择一个信号,然后按下 Trigger signal 按钮,来选中触发信号。如果一个信号被选中为触发,那么字母"T"会出现在模块名字的左边。注意,这个标记只在触发源被置为信号触发选项时,才会出现。在对话框的下端,有不同的选项可以使用:

(1)Source——触发源。有两种选择:mannual 或者 signal。选择 mannual,在外部模式控制面板的 Arm trigger 按钮一被按下,就开始记录数据。选择 signal 则告诉外部模式在选中的触发信号满足触发条件(按指定的方向滑过触发门限)时就开始记录数据。

(2)Duration——持续时间。这个实际也就决定了外部模式在设定的触发事件发生后,记录数据持续的基准速率时间步的数目。例如,模型中最快的速率是 1 s,对于频率为 1Hz 的信号,如果记录的 Duration 设为 10 s,于是外部模式将会收集 10 个采样。如果 2Hz 的信号被记录,在同样的持续时间下,那么 5 个采样被收集。

(3)Mode——模式。也有两种选择:normal 或者 one - shot。在 normal 模式,外部模式在每个出发事件发生后自动重新准备触发器。而在 one - shot 模式,外部模式在用户没准备好触发器一次,就只收集一个缓存的数据。

(4)Delay——延迟时间。用户可以在触发器设置一个延迟,其数值是基本速率的仿真步数目,可以为正也可以为负,表示数据收集相对于触发发生的延迟时间。一个负的时延就对应着前面的一个触发。

(5)Arm when connect to target——当连接到目标时准备好触发器。这个检查框就告诉外部模式在用户连接到目标时,就自动准备好触发器。一旦触发器被准备好,外部模式就监视触发信号是否具有指定的触发条件。

在触发信号选择好之后,触发信号面板将变为可用状态,用户可以在它上面定义触发条件,以及设置信号所处的 Port(端口)和 Element(元素),因为前面选择触发信号,是根据所属模块来标记的。缺省情况下,所设定的触发模块的第一个端口的任何元素都能够导致触发器被触发。用户可以调整 Port 值和触发面板右端的 Element 域来改变这些行为。Port 域可以是一个数字或者关键字 last,而 Element 域则能接收一个数字或者关键字 any 和 last。

触发信号面板还可以设置触发信号触发条件,供选择的选项有:

(1)Direction——触发方向。它包括:rising(上升)、falling(下降)或者 any(任何)。触发信号只有在按设定的方向滑过门限值才会产生激活触发器,也可以理解为触发事件的类型,它们和触发子系统里的定义是一样的。

(2)Level——说明触发信号的门限值。触发信号只有在按 Direction 所设定的方向滑过这个值时才会激活触发器。缺省情况下,level 值设为 0,如果它被设置为 1,那么,将使触发器在出发信号按设定的方向滑过 1 时才触发。

(3)Hold - off——只在 normal 模式下有用。其单位是基准的时间步,它表示一次触发事件和触发器准备好的时间间隔。

3. 数据记录

这个窗口支持外部模式的以下特征:

(1)Direction notes——目录层次注释。设置它,可以通过 Edit directory note 按钮打开 MATLAB 编辑器(M 文件),要保存在特定目录的注释就可以放在编辑窗口里。

(2)File notes——文件层次注释。设置它,可以通过按 Edit file note 按钮打开文件查找窗口。缺省下选中的文件是最近写过的文件。选择任何 MAT 文件打开一个编辑窗口,然后就可以添加或者编辑注释,并保存为自己的文件。

(3)Data archiving——自动的数据记录和存储。按 Enable Archiving 按钮就激活了外部模式的自动数据档案特性。为了了解数据档案特性工作的原理,读者有必要知道在档案特性被禁止是数据获取的工作机制。这有两种情况:one - shot 和 normal 模式。在 one - shot 模式,在一个触发事件发生后,每一个被选择为触发器的 Scope 模块只会在仿真快要结束时,才写数据到工作空间。如果仿真过程中,还有另外一个 one - shot 事件被触发,那么工组空间中已存在的数据将会被擦除重写。在 normal 模式,外部模式在每一个触发事件发生后,就自动地重新准备触发器。所以,读者可以把 normal 模式看成一串的 one - shot,其中的每一个 one - shot,除了最后一个之外,都会引起一个即时结果。因为触发器可以在任何时候触发,写即时结果到工作空间会导致对工作空间变量不可预测的重写。所以,缺省的行为是在最后一个 one - shot 触发后才把结果写入工作空间。而即时结果被摒弃。如果用户知道在触发器里有足够的时间间隔来监视即时结果,那么用户就可以选中 Write to intermediate workspace 检查框来重新定义这种行为。注意这个选项不保护工作空间数据不被后面的触发器重写。

Exteranl Data Archiving 对话框的选项支持保存数据自动写入(包括即时结果)到磁盘。数据档案提供下面的设置:

• Directory——设定保存数据的目录,如果选择了 Increment directory when trigger armed 检查框,外部模式就会自动添加一个后缀到目录名上。

• File——指定数据保存的文件名,在 Increment file after one - shot 被选中后,外部模式也会自动添加一个后缀到文件名后。

• Increment directory when trigger armed——外部模式在每次 Arm trigger 按钮被按下时,都会自动使用一个不同的目录来存放数据保存文件。目录的名称是通过在设定的目录后添加一个按升序增加的数字来命名(这种文件称为增序文件)的,例如,dirname1,dirname2,等。

• Append file suffix to variable names——无论何时的外部模式增加文件名的序号,每一个文件保存的变量都会被加上所处文件的序号作为后缀。例如,文 file_1,file_2 文件里的数据变量 xdata 分别为 xdata_1,xdata_2,于是每一个文件内的变量名在整个工作空间里就是单一的。这主要是为了便于在 MATLAB 比较变量。如果不是单一的,那么在分析时,新文件里的变量就会在 MATLAB 工作空间,覆盖前面文件里的变量。图 9 - 22 显示了在 archive 被使能下 External Data Archiving 对话框的样子。



图 9 - 22 External Data Archiving 对话框

9.4.4 外部模式的 TCP/IP

这一节来介绍如何使用 RTW 实现基于 TCP/IP 的客户机/服务器的计算模式。在读者的目标系统支持 TCP/IP 的前提下，读者可以使用 RTW 生成代码提供的基于 Socked 的外部模式来实现。

低层次的传输层负责消息的物理传输，Simulink 和模型代码和这一层都是独立的，它们被分隔在分离的代码模块中，例如，格式、发送、接受消息以及数据包。

使用 Simulink 外部模式，必须进行下面的操作：

(1)在外部目标接口对话框，设定外部接口 MEX - 文件的名称。缺省情况下，这个文件是 ext _ comm.。

(2)配置模板 makefile，以便它能为 TCP/IP 服务器代码链接到更合适的源代码，并且定义建立生成代码所必需的编译器标识。

(3)建立外部程序。

(4)运行外部程序。

(5)将 Simulink 设为外部模式，并且连接到目标。

介绍外部模式的 TCP/IP 实现，也将遵循这样的步骤。图 9 - 23 显示了基于 TCP/IP 的外部模式实现的结构。

建立外部程序的第一件事是设置外部接口 MEX - 文件，它的缺省设置是 ext _ comm，它实现了基于 TCP/IP 的通信。Ext _ comm 具有三个额外的参量，读者可以在外部接口文件对话框的参数对话框设置这些参量。它们如下所述。

(1)目标网络名：运行外部程序的计算机的网路名称。缺省情况下，它的取值为运行 Simulink 的计算机的名称，这就是宿主和目标相同的外部模式。这个名称可以是一个用单引号限定的字符串，例如‘myPuter’，也可以是单引号限定的 IP 地址，例如‘202.113.26.16’。

(2)详细等级：控制数据传送过程中所显示信息的详细等级。取值为 0 或者 1，它们的定义如下：

0——无信息；

1——详细的信息。

(3)TCP/IP 服务器端口数：缺省值是 17725。读者在必要时，可以把它设为一个处于

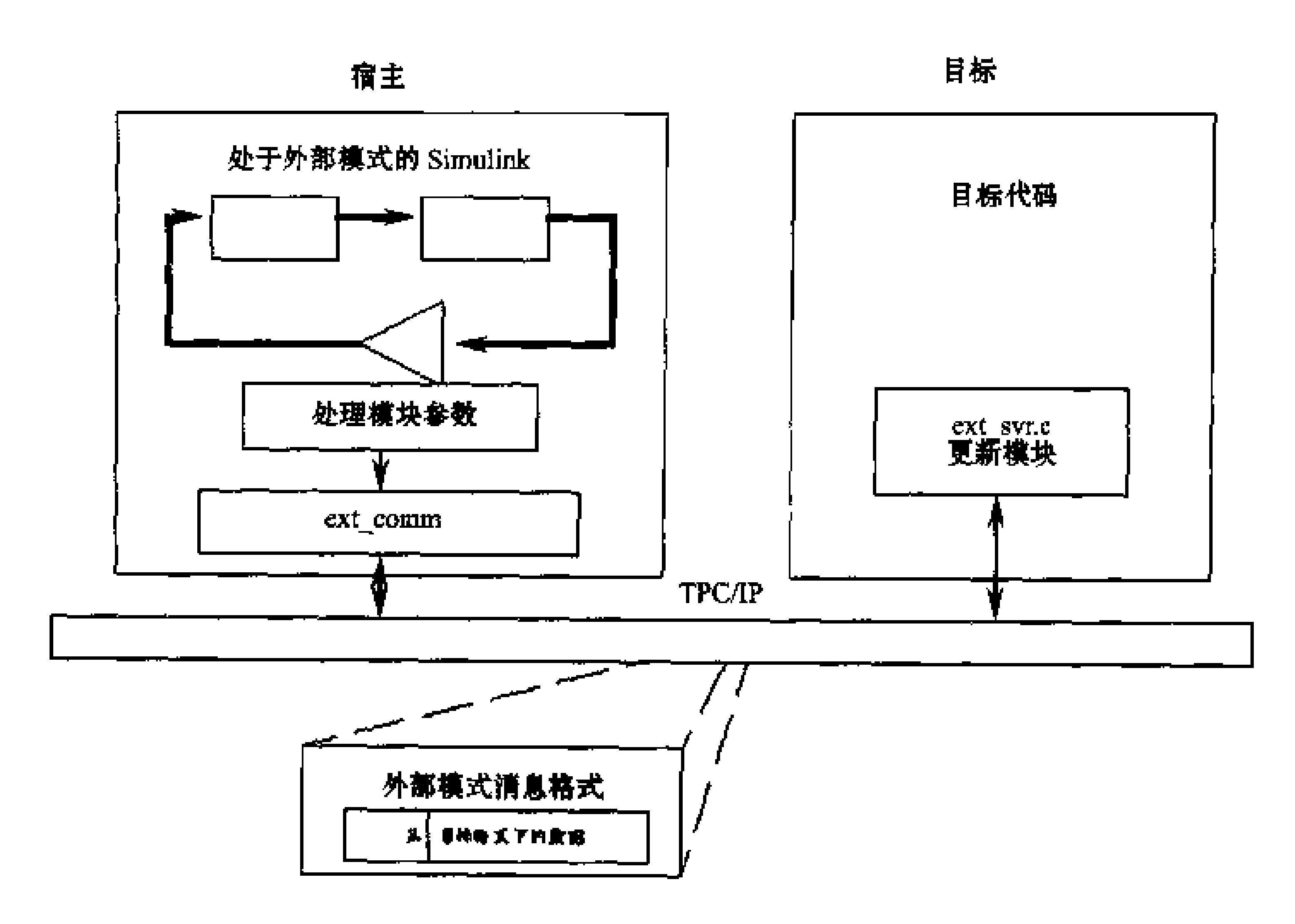


图 9-23 基于 TCP/IP 的外部模式实现的结构

256 和 655356 之间的值,来避免端口冲突。

读者在设定上面的选项时,必须按照次序进行。例如,如果读者想设置信息的详细等级(第二个参量),那么你必须要同时设定目标机器名(第一个参量)。

接下去的操作和前面讲的差不多。为了能使外部模式代码生成,读者要在 Real - Time Workshop 页的 target - specific code generation option 面板选中外部代码检查框。

外部模式代码生成之后,就要运行外部模式代码,读者可以使用前面的两种方式:在 MS - DOS 命令行输入 modelname - tf inf - w,或者直接在 MATLAB 命令窗口输入

! *modelname - tf inf - w &*

关于命令的参量说明请看 9.3.2 节。

如果 Simulink 模块图表和外部程序不匹配,Simulink 显示一个错误对话框告诉用户检查不匹配,可能在生成代码后,模型已经被修改过。这种情况的解决方式,是重新根据新的模块图表建立外部程序。如果外部程序不运行,Simulink 也会给出一个错误提示,表明无法连接到外部程序。

9.5 RTW 代码库

为了方便用户构建实时系统模型和生成代码,RTW 提供了一个函数库,读者可以用 SIMULINK 库浏览器看到这个库,如图 9-24 所示。

从图中读者可以看到,RTW 库有五个子库组成,它们分别是:

(1)Custom Code Library——模块集合,它允许用户将自定义的代码插入到生成的源代码和用户模型相关的函数;

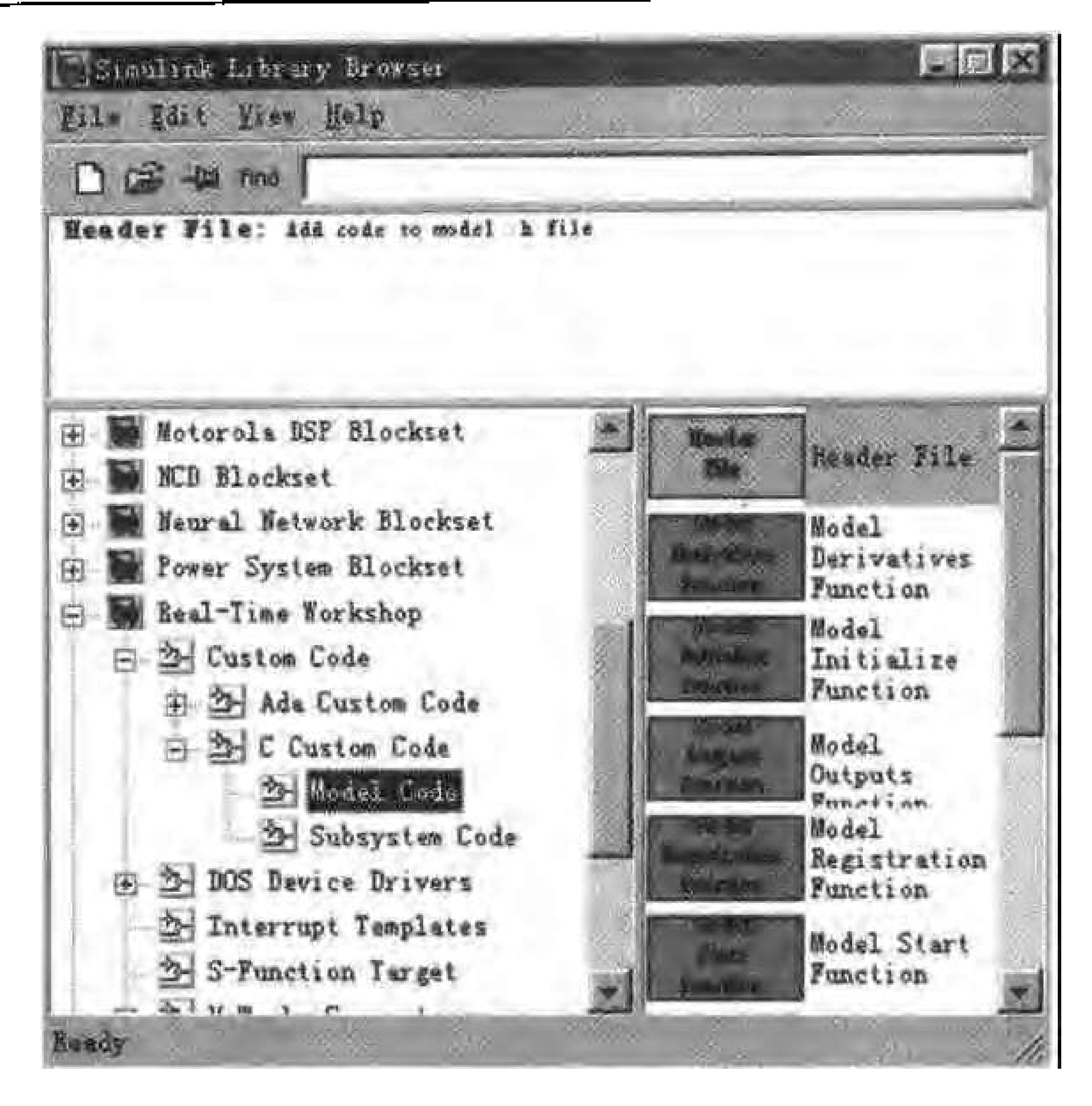


图 9-24　RTW 库

(2)DOS Device Drivers——为了 DOS 目标设计的模块;

(3)Interrupt Templates——用户可以把它作为模板生成自己的异步中断模块集合;

(4)S-Funcion Target——这个模块用于和 Real Time Workshop S-函数代码格式结合使用;

(5)VxWorks Support——支持 VxWorks(Tornado)的模块集合。

9.5.1　Custom Code Library (自定义代码库)

自定义代码库的作用是为用户提供一些可以在 RTW 生成的代码里放置自己的代码(可以用 C 也可以用 Ada 编写)的模块。对于 C 和 Ada 自定义代码库,都有两个自定义代码子库:

(1)Custom Model Code (自定义模型代码);

(2)Custom Subsystem Code(自定义子系统代码)。

这两个子库包含的模块,用户都可以在里面放置自己的代码,这些模块分别指向特定的模型文件和子系统。

1. 自定义模型代码

自定义模型代码包含了 10 个模块,它们的编辑域使用户可以在生成的模型文件和函数中插入自定义代码。读者可以打开这个子库来观看这些模块,在子库中用右键上的命令打开窗口,如图 9-25 所示。其中在第一行的四个模块的作用分别是在下面的四个文件的顶部和底部插入自定义代码。描述如下:

(1) model.h——Header File 模块;

(2) model.prm——Parameter File 模块;

(3) model.c——SourceFile 模块;

(4) model.reg——Registration File 模块。

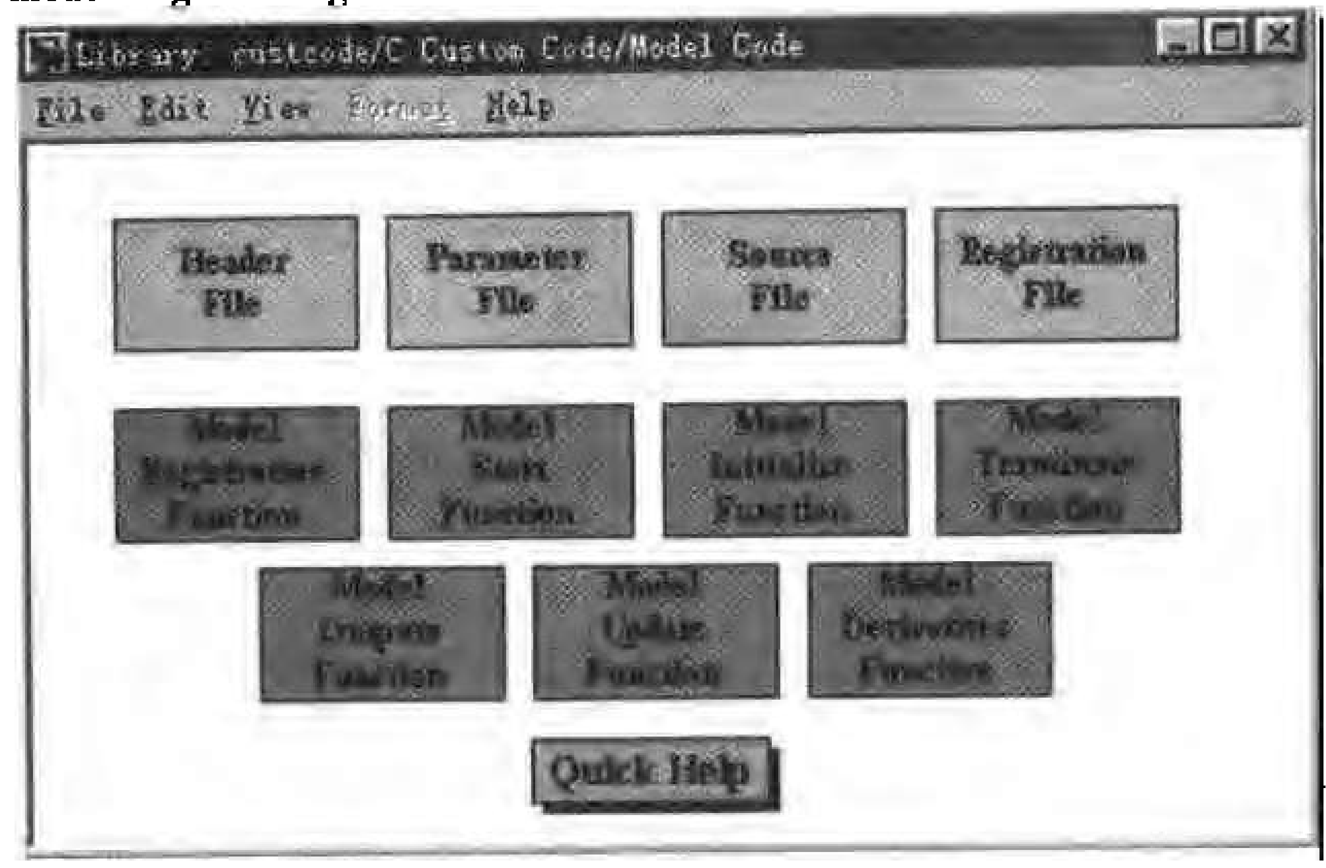


图 9-25　自定义模型代码库

而第二行和第三行的七个模块,用户可以在它们的编辑域编辑要插入在下而所述的模型函数的代码:

(1) MdlRegistration——MdlRegistiation Function 模块;

(2) MdlStart——MdlStart Function 模块;

(3) MdlInitialize——MdlInitialize Function 模块;

(4) MdlTerminate——MdlTerminate Function 模块;

(5) MdlOutputs——MdlOutputs Function 模块;

(6) MdlUpdate——MdlUpdate Function 模块;

(7) MdlDerivatives——MdlDerivatives Function 模块。

2. Custom Subsystem Code(自定义子系统代码)

Custom Subsystem Code 库提供了八个模块,如图 9-26 所示,用户可以使用它们把代码插入到系统函数中。

这八个模块分别是:

(1) Subsystem Start 模块;

(2) Subsystem Initialize 模块;

(3) Subsystem Terminate 模块;

(4) Subsystem Enable 模块;

(5) Subsystem Disable 模块;

(6) Subsystem Outputs 模块;

(7) Subsystem Update 模块;

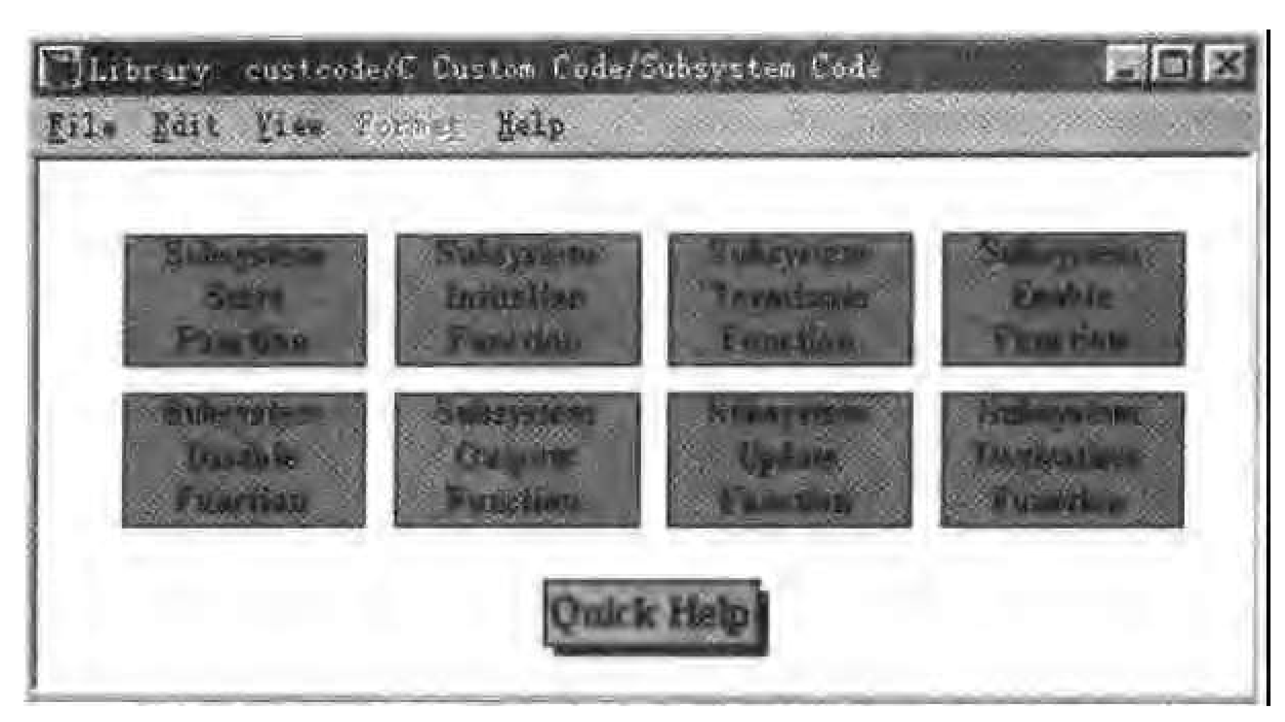


图 9 - 26 Custom Subsystem Code 子库

(8) Subsystem Derivatives 模块。

这些子系统模块所在的位置决定了里面的自定义代码在模型代码中的位置。换言之,代码对所选择的子系统是本地的。例如,Subsystem Outputs 模块在模块处于模型的顶层时,就把代码放置在 MdlOutputs 函数中,但是如果它处于一个使能子系统内,那么代码就被放置在这个系统的输出函数中。

对于一个触发子系统,其中个输出函数的次序为:

(1) Output entry;

(2) Output exit;

(3) Update entry;

(4) Update exit code。

9.5.2 使用自定义代码模块示例

下面将举一个例子来演示如何使用自定义代码模块。这个例子将通过 MdlStart Function 模块将代码插入到 MdlStart 函数中,图 9 - 27 是所用的模块图,其中插入了一个 MdlStart Function 模块。

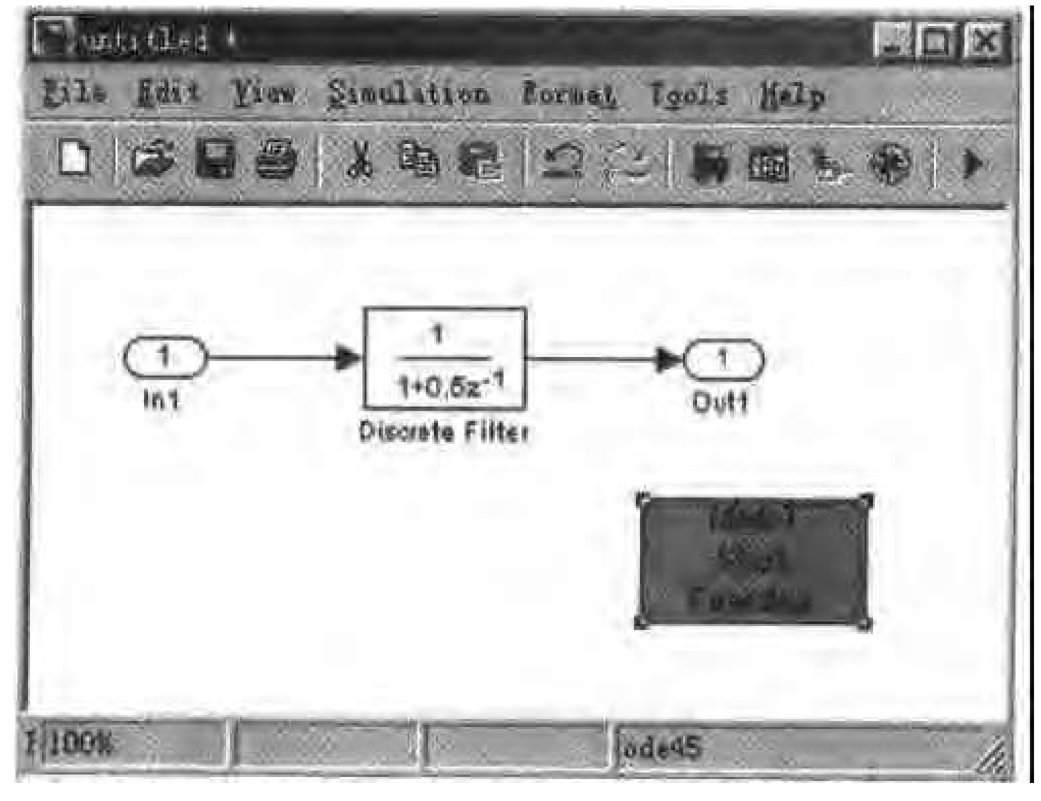


图 9 - 27 自定义代码模块使用示例模型

双击 MdlStart Function 模块,将打开如图 9-28 所示的对话框,请按对话框所示,在相应的编辑域里输入代码。当然,读者可以任意地在编辑域输入代码。

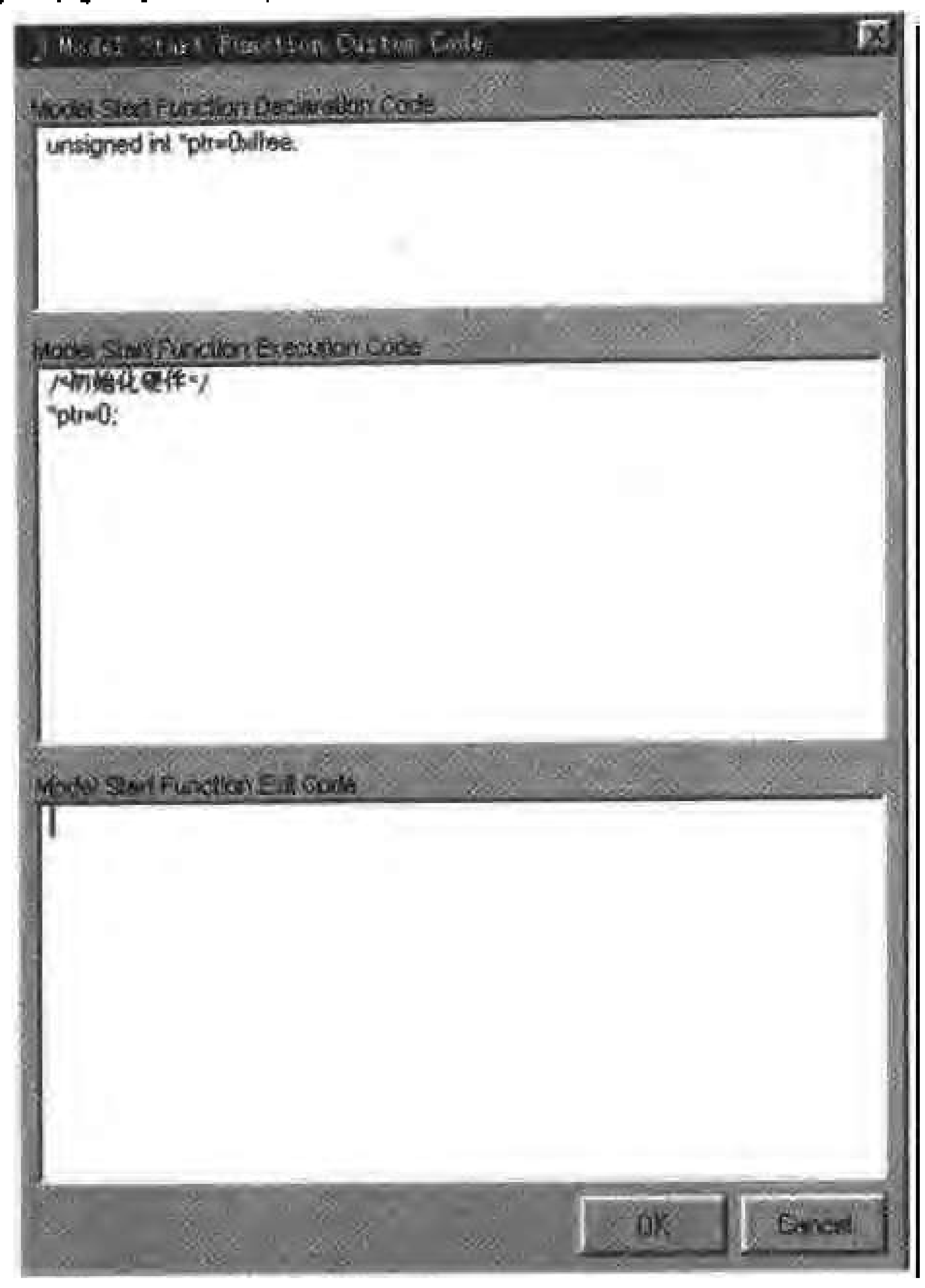


图 9-28 在 Model Start Function 对话框输入代码

在声明代码编辑框里输入的代码为:"unsigned int ptr = 0xFFEE ;",在执行代码编辑框里输入的两行代码为:"/ * 初始化硬件 * / "和" * ptr = 0"。

于是,根据本例生成的代码的 MdlStart 函数就包含下面的代码:

```
void MdlStart (void)
{
/* user code (Start function Header) */
/* System : <Root> */
unsigned int ptr = 0xFFEE ;
/* user code (Start function Body) */
/* System : <Root> */
/* 初始化硬件 */
 *ptr = 0 ;
/* state initializetion */
/* DiscreteFilter Block : <Root>/Discrete Filter */
rtX.d.Discrete_Filter = 0.0 ;
```

```
}
```

从中可见,在 MdlStart 函数模块对话框里输入的代码被直接嵌入到生成的代码中去了。

RTW 还提供一个中断模板库,支持用户对同步或者异步事件的处理,包括中断服务方法(ISRs)。使用库里的模块,读者就可以建立能够处理异步事件的模型,这些事件包括硬件产生的中断,和异步读写操作等等。这方面的信息,已经超出本书的范围,有兴趣的读者可以参考 RTW 的帮助文档。

第10章　用S-函数扩展Simulink

通过前面几章的学习，读者对用Simulink建模的基本思路应该有了一个比较清晰的认识，Simulink为用户提供了许许多多的内置库模块，正如MATLAB语言的内置函数一样，用户的主要工作就是将所要解决的问题映射到由这些模块并搭在一起的系统。为了方便用户更有条理、便捷地管理模型，Simulink提供了许多的虚拟模块来建立模型。对于那些经常使用的模块组合(子系统)，Simulink允许用户把它们建成子系统，并封装成能够重复使用的库模块，这样就减少许多不必要的重复劳动。可以说，这种封装方式也是一种对Simulink进行扩展的方式，但是它是基于Simulink原来提供的内置模块的。在这一章里，读者学习另外一种扩展Simulink的强大机制——S-函数，它是System function——系统函数的简称。

10.1　S-函数综述

10.1.1　什么是S-函数

S-函数是一个动态系统的计算机语言描述，在MATLAB里，用户可以选择用MATLAB语言还是C语言来编写S-函数，MATLAB6.0还支持使用C++来编写S-函数。前者在本书里，不妨称之为M文件S-函数，而后者则取名叫C-MEX文件S-函数。对于后者，读者也许会很陌生，这里姑且放在一边，读者只要记住C-MEX文件是C语言编写的S-函数在MATLAB被编译成的一种可执行文件就行了。

S-函数采用一种特殊的调用语法，使函数和Simulink方程解法器进行交互，这种交互与在解法器和Simulink间发生的交互十分类似。S-函数的形式十分通用，它能够支持连续系统、离散系统和混合系统，可以说几乎所有的Simulink模型都可以用S-函数来描述。

请读者先看一个简单的S-函数例程。

这个文件读者无需手动输入，只要用MATLAB Edit窗口打开你的机器的MATLAB安装目录下的toolbox \ Simulink \ bloocks目录里的timestwo文件即可。它是一个把输入信号乘以2倍的模块的MATLAB语言描述。而这用Simulink的内置模块实现，只需要一个gain模块就足够了。

```
function [sys,x0,str,ts] = timestwo(t,x,u,flag)
%TIMESTWO S-function whose output is two times its input.
%   This M-file illustrates how to construct an M-file S-function that
%   computes an output value based upon its input. The output of this
```

```
%   S-function is two times the input value:
%
%     y = 2 * u;
%
%   See sfuntmpl.m for a general S-function template.
%
%   See also SFUNTMPL.
%   Copyright 1990-2000 The MathWorks, Inc.
%   $Revision: 1.5 $

%Dispatch the flag. The switch function controls the calls to
%S-function routines at each simulation stage of the S-function.
switch flag,
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  % Initialize the states, sample times, and state ordering strings.
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes;

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  % Return the outputs of the S-function block.
  case 3
    sys = mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%
  % There are no termination tasks (flag = 9) to be handled.
  % Also, there are no continuous or discrete states,
  % so flags 1,2, and 4 are not used, so return an emptyu
  % matrix
  case { 1, 2, 4, 9 }
    sys = [];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags (error handling)%
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Return an error message for unhandled flag values.
    otherwise
      error(['Unhandled flag = ',num2str(flag)]);

end
% end timestwo

% ==============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
% ==============================================================================
function [sys,x0,str,ts] = mdlInitializeSizes()

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = -1;  % dynamically sized
sizes.NumInputs      = -1;  % dynamically sized
sizes.DirFeedthrough = 1;   % has direct feedthrough
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
str = [];
x0 = [];
ts = [-1 0];     % inherited sample time

% end mdlInitializeSizes

% ==============================================================================
% mdlOutputs
% Return the output vector for the S-function
% ==============================================================================
function sys = mdlOutputs(t,x,u)
```

```
sys = u * 2;

% end mdlOutputs
```

请读者姑且把编写 S - 函数的种种疑问搁置在一边，就假定 S - 函数已经编好了，先来看看 S - 函数是如何被合并到 Simulink 模型里的。这也就是如何在 Simulink 里调用 S - 函数的问题。为此，就需要使用 functions & tables 子库里的 S - function 模块。

如图 10 - 1 所示，请读者在 S - 函数的对话框的 S - function name 编辑框内输入要调用的函数名，这里是 timestwo，至于 parameters 参数在这个例子里就不用填写了。设置完参数的 S - function 模块就如图 10 - 2 显示的一样。下面就可以使用这个模块了，读者不妨建立一个简单的模型来测试一下它的作用，例如将 sinewave 模块作为 S - 函数模块的输入，而把它的输出接到一个 scope 模块显示。如图 10 - 3 所示。

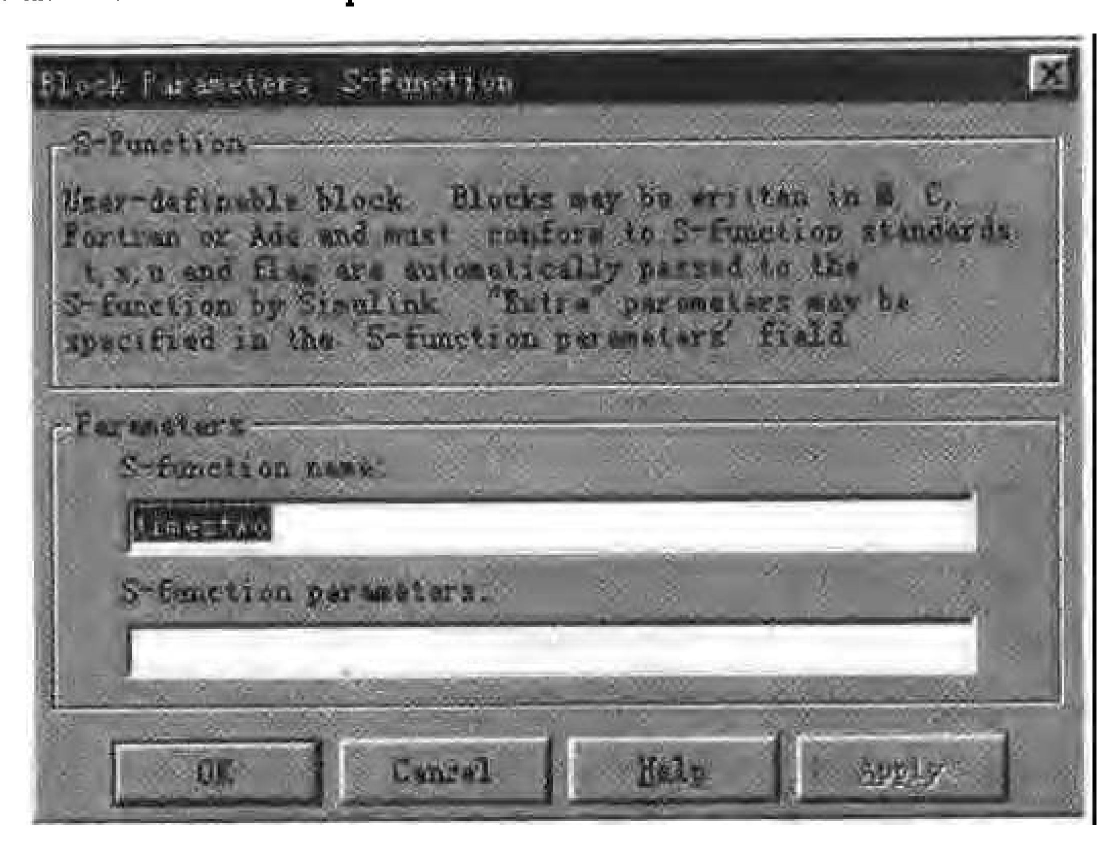


图 10 - 1　S - Function 函数设置

但是上面的模块在被双击后出现的对话框仍然是 S - 函数的对话框，而且有时候用户编写的 S - 函数还可能需要传递参数进去，这时再使用这个对话框就不太方便了。比较好的方法，是对 S - function 模块进行封装，自定义 S - 函数的参数对话框，使封装后的 S - 函数模块表现得与内置模块一样。

这里就有一个问题，什么样的参数才是额外参数呢?

在前面的 S - 函数的源代码例子里，S - 函数的说明语句为

function [sys, x0, str, ts] = timestwo (t, x, u, flag)

上面的这些输入参数 t, x, u 和 flag 是编写 S - 函数所必需的输入参量，这些参数在设置 S - function 模块参数时不再填写在额外参数编辑框里，也就是说它们由 Simulink 解法器来自动传给 S - 函数。除了它们之外的任何参数都是额外参数，需要用户从外部传值给它们。因此，额外参数必须按照函数声明的顺序在 S - 函数参数对话框的 parameter 编辑框里进行说明。例如，要定义一个具有额外参量 a 和 b 的 S - 函数 sexample，就必须如图 10 - 4 所示去设置模块的参数编辑框。

function [sys, x0, str, ts] = sexample (t, x, u, flag, a, b)

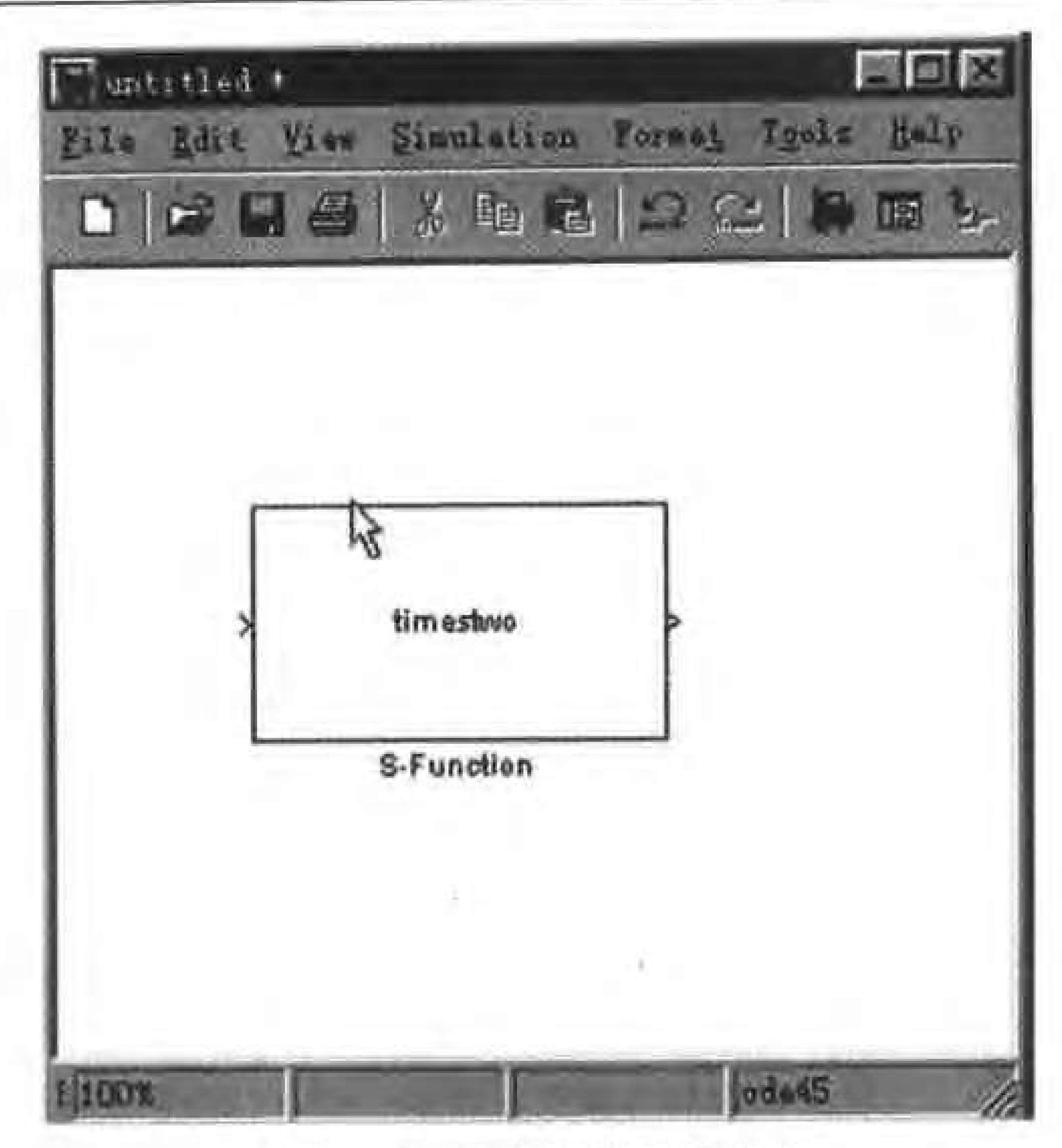


图 10－2　S－函数设置好的情况

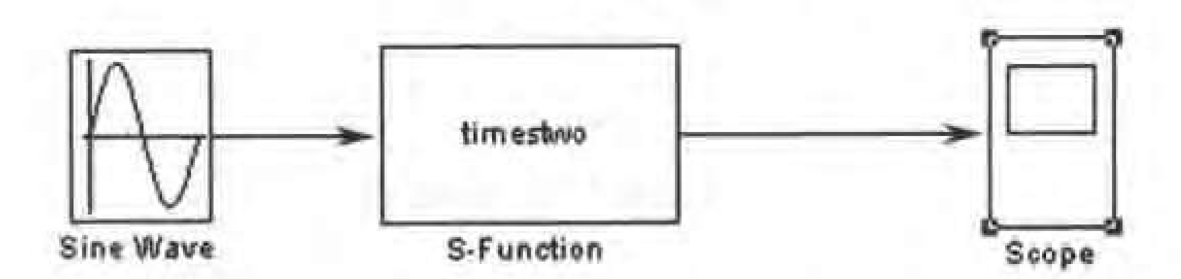


图 10－3　一个简单的测试系统

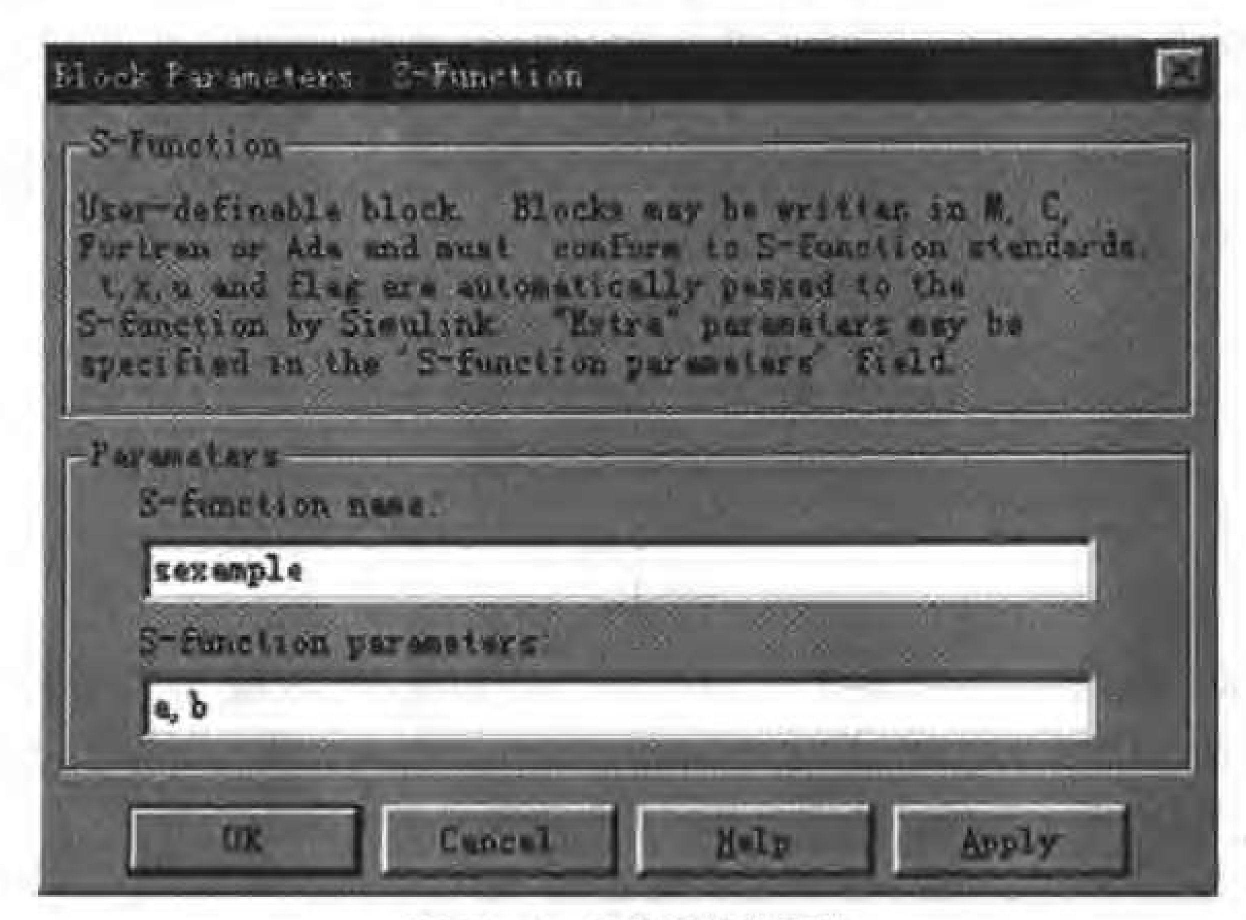


图 10－4　额外参数的设置

读者可以看到，在 scxample 函数里，a，b 就是额外参数，它们使用时需要用户定义，所以要在编辑框里说明，而说明的顺序就是函数文件里说明的顺序。然后再运用与普通的子系统封装一样的方法来对 S－函数模块进行封装，自定义它的模块图和参数设置对话框。

前面说过，S－函数最广泛的用途是定制用户自己的 Simulink 模块，更具体地讲，它的作用体现在以下几点：

(1) 用户可以用它来创建新的通用性的 Simulink 模块；

(2) 将已存在的 C 代码合并到仿真中；

(3) 将一个系统描述成一个数学方程；

(4) 便于使用图形仿真。

使用 S－函数的一个好处就是，用户可以建立能多次使用的通用模块，而模块的每个示例可以具有不同的参数值。

10.1.2 S－函数如何工作

在具体讲述 S－函数的工作原理之前，还是先来回顾一下前面讲过的 Simulink 模块的工作模型。

Simulink 中的每一个模块都有三个基本的元素：输入向量、状态向量和输出向量，本书分别用 $\boldsymbol{u}$、$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 来标记它们。图 10－5 反映了三个元素之间的关系。

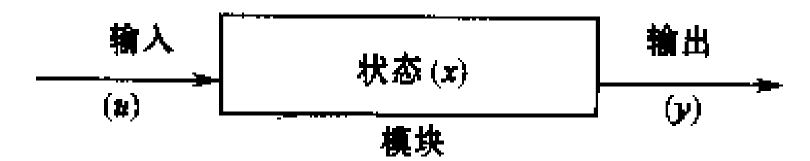


图 10－5 Simulink 模块的基本模型

在上面的三个基本元素中，状态向量无疑是最重要的，也是最灵活的一个概念。何谓状态呢？读者首先想到的可能是线性理论里的状态方程里的状态，在那里，状态被定义为一些微分量。可以说那里的状态只是状态这个概念内涵的一部分。不仅仅只是工程系统才存在状态，事实上，状态这个概念在非工程系统也能找到它的对应物。

在 Simulink 里状态向量可以分为连续状态、离散状态，或者是两者的结合。输入、输出、状态这三个量的关系可以用下面的方程来反映。

$$\boldsymbol{y} = f_0(t, \boldsymbol{u}, \boldsymbol{x})$$

$$\boldsymbol{x}_{d_{k+1}} = f_u(t, \boldsymbol{u}, \boldsymbol{x})$$

$$\boldsymbol{x}_c' = f_d(t, \boldsymbol{u}, \boldsymbol{x})$$

其中

$$\boldsymbol{x} = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$$

Simulink 在仿真时把上面的这些方程对应为不同的仿真阶段，它们分别是计算模块的输出、更新它的离散状态和计算连续状态的微分。在仿真的开始和结束，还包括初始化和结束任务两个阶段。在每一个阶段，Simulink 重复地对模块进行调用。

关于仿真，读者已经接触到了好几个概念，仿真步(Simulation Step)、仿真阶段(Simulation Stage)。这里还要增加一个概念：仿真循环(Simulation Loop)。一个仿真就是由仿

真阶段按一定顺序组成的执行序列。对于每一个模块，经过一次仿真循环就是一个仿真步，而在同一个仿真时间步，模型中各个模块大的仿真步按照事先排列好的次序依次执行。这个过程可以用图 10－6 来表示。

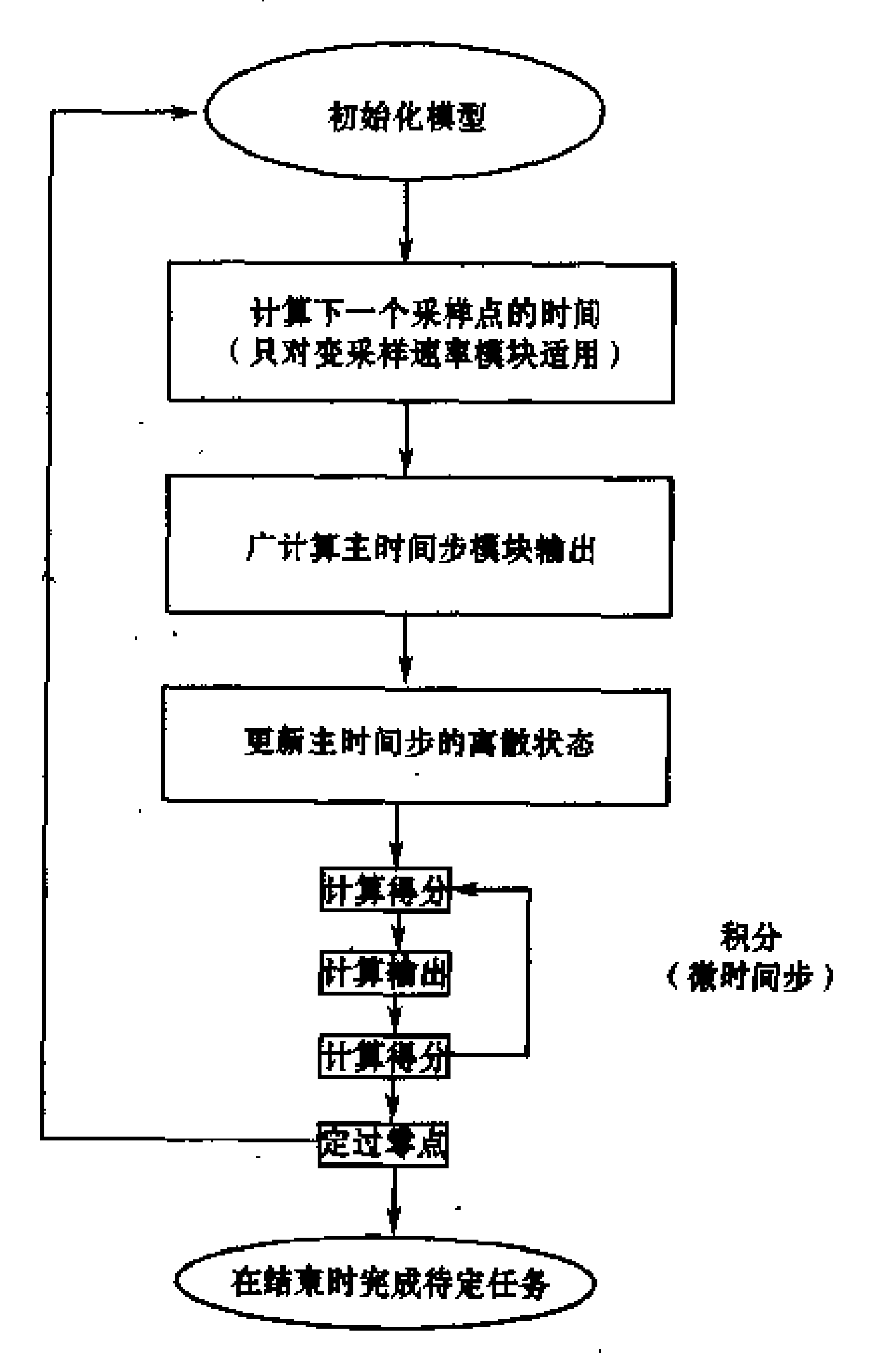


图 10－6　仿真执行流程图

在调用模型中的 S－函数时，Simulink 会调用用户定义的的 S－函数方法（S－function routines），来实现每个仿真阶段要完成的任务。这些任务包括：

（1）初始化阶段。它是仿真进行的第一步，先于第一个仿真循环，它初始化 S－函数，在这个阶段，Simulink 完成下面的工作：

- 初始化 SimStrict——包含 S－函数信息的数据结构；
- 确定输入、输出端口的数目和大小；
- 确定模块的采样时间；
- 分配内存和 sizes 数组。

（2）计算下一个采样点的时间。如果模型使用变步长解法器，那么就需要在当前仿真步确定下一个采样点的时刻，也即下一个仿真步的大小。

（3）计算当前主仿真步的输出。在这个调用完成之后，模块的所有输出端口都对当前的仿真步有效，也就是说在一个模块的输出被更新之后，它才能作为其他模块的有效输入，去影响那些模块的行为。

(4)更新模块当前主时间步的离散状态。在这个仿真阶段中,所有的模块都要进行每个时间步一次的活动——为当前时间步的仿真循环更新离散状态。

(5)积分。这个阶段只有偶模块具有连续状态的非采样过零点时,才会存在。如果它是 S－函数,那么 Simulink 按最小时间步来调用 S－函数的输出和微分 S－函数方法。如果 S－函数具有非采样过零点(只有 CMEX 函数才有可能存在这种情况),类似地,Simulink 按最小时间步来调用 S－函数中的输出和过零点部分,以便 Simulink 确定过零点的位置。

10.1.3 M 文件和 C MEX 文件 S－函数综述

在 M 文件 S－函数中,S－函数方法是用 M 文件函数来实现的,而在 C MEX 文件 S－函数中,它们用 C 函数来实现。M 文件 S－函数能使用的 S－函数方法都可以在 C MEX 文件 S－函数中找到对等的函数。但是反过来不成立,Simulink 为 C MEX S－函数提供了更多的方法。在 M 文件和 C MEX 文件 S－函数中都存在的方法,在两者文件里具有相同的名称。

对于 M 文件 S－函数,Simulink 通过传递一个 flag 参量给 S－函数,告诉 S－函数当前所处的仿真阶段,以便相应的子函数。于是,在写 M 文件 S－函数,用户只需用 MATLAB 语言为每个 flag 对应的 S－函数方法来编写代码即可。而在 C MEX 文件 S－函数中,Simulink 直接调用相应的 S－函数方法,而不是通过额外的参量(见后面的 C MEX 文件编写)。表 10－1 列出了仿真阶段各自对应的 S－函数方法和 M 文件 S－函数中与它们相对应的 flag 值。

表 10－1　各个仿真阶段对应的方法

仿真阶段	S－函数方法	Flag(M 文件 S－函数)
初始化	mdlInitializesSizes	0
下一个采样点的计算(附加)	MdlGetTimeOfNextVarHit	4
输出值的计算	MdlOutputs	3
更新离散状态	MdlUpdate	2
微分的计算	mdlDerovatoves	1
仿真任务的结束	MdlTerminate	0

C MEX S－函数里,S－函数方法的名称必须和表中的一样。但是在 M 文件 S－函数中,S－函数方法(子函数)的名称不一定非得是表中的名称。读者可以用 switch 语句为每个 flag 值规定与它对应的 M 文件自函数的名称。为了便于用户编写 S－函数,Simulink 提供了一个名为 sfuntmpl.m 的模板文件,读者可以在 matlabroot/toolbox/Simulink /blocks 里找到它。使用模板文件写 M 文件函数,用户只需把自己的代码放到相应的 S－函数方法中去即可。而对于 C MEX S－函数,Simulink 同样提供了一个模板文件,它的名称是 sfuntmpl.c,所处的目录是 Simulink/src。

10.1.4 S－函数概念

理解下面的这些关键概念,将有助于读者正确地建立 S－函数:

(1) 直接馈入；

(2) 宽度动态可变的输入信号；

(3) 设置采样时间和偏移。

1. 直接馈入

正如前面所讲述过的，直接馈入指模块或者采样时间(变速率模型)直接由它的某个输入端口来控制。判断一个 S－函数是否具有直接馈入的标准有：

(1)输出函数(MdlOutputs 或者 flag = 3)是一个输入参数包含 u(从 S－函数的角度)的函数，也就是计算系统输出的方程里包含变量 u 时，这个 S－函数就被认为是具有直接馈入。这一点和前面的论述是一致的。

(2)如果 S－函数是一个采样时间可变的 S－函数，并且下一个采样点的计算要求用到输入参数 u 时，也认为它是具有直接馈入的。

一个具有直接馈入的例子就是 $y = k * x$ 这个公式，其中的 k 是增益，u 是输入信号，y 是输出。而不是有直接馈入的例子是下面公式表达的简单积分算法。

输出：$y = x$

微分：$x' = u$

其中，x 是状态值，而 x' 是状态对时间的微分，y 表示输出。

也许初学者会问，当传入的 u 是以前的输入值时，输出的函数包含 u 也是直接馈入吗？其实这种问题是不存在的，因为 Simulink 传给 S－函数的 u 都是当前时刻的输入值。如果 S－函数的某个子函数要用到以前的输入，那也只能用状态变量 x 来传递，当然，要先把输入存储到状态向量。因此，前面说的规则和模块的直接馈入的定义是不矛盾的。

2. 宽度动态可变的输入信号

S－函数可以被写成支持任意输入宽度。这种情况下，输入信号的实际宽度可以在仿真开始阶段，通过 size 或者 length 等函数来求出输入向量的宽度。然后，就可以利用这个宽度来估计连续状态数目、离散状态数目和输出向量的宽度。

关于 S－函数的输入，还有一点需要让读者注意。我们知道 Simulink 的内置模块的输入往往是分成多个端口，而每个端口又可以分别由若干个元素组成。这个特性的实现，在 M 文件 S－函数和 C MEX S－函数是有差别的。粗略地说，M 文件 S－函数对端口不加区分，它把所有端口合成一个输入向量；而 C MEX S－函数允许用户设置输入的端口数，然后再设置每个端口的元素个数。更详细的描述，读者可以在关于这两种 S－函数的编写的具体介绍中找到。

3. 设置采样时间和偏移

M 文件和 C MEX S－函数都允许用户十分方便地设定 S－函数被调用地时间，也就是设置采样时刻。Simulink 为采样时间提供了下面的几种选择：

(1)连续的采样时间——适用于具有连续状态和非采样过零点的 S－函数，这种 S－函数输出按最小的时间步改变。

(2)连续的，但是值最小的仿真步具有固定值的采样时间——适用于需要在每一个主仿真时间步执行，但是在最小仿真步内值不改变的 S－函数。

(3)离散采样时间——如果 S－函数的行为的发生具有离散时间间隔的函数，用户可以定义一个采样时间来规定 Simulink 何时调用函数。而且用户还可以定义一个延迟时

间 offset 来延迟采样点,注意 offset 的值不能超过采样周期,也就是相邻采样点的时间间隔。

一个采样点对应的时间值由下面公式决定:

$$TimeHit = (n \times period) + offset$$

其中,n 是一个整数,它表示当前的仿真步,起始值总为 1。

如果用户定义了一个离散采样时间,那么 Simulink 就会在所定义的每个采样点调用 S-函数的 MdlOutput 和 MdlUpdate 方法。

(4)可变采样时间——相邻采样点的时间间隔可变的离散采样时间。在这种采样时间制式下,S-函数要在每一个仿真步开始,计算下一个采样点的时刻。

(5)继承的采样时间——有时候,一个 S-函数的模块自身没有特定的采样时间,而它属于连续还是离散采样时间,完全取决于系统中的其他模块的采样时间。编写描述这种特性的 S-函数模块时,可以把采样时间设定为继承。例如,gain 模块就是一个继承输入信号的采样时间例子。一个模块可以从以下几种方式来继承采样时间:

- 继承驱动模块的采样时间;
- 继承目的模块的采样时间;
- 继承系统中的最快采样时间。

把采样时间设为继承的,要把 sample time 的值置为 -1。S-函数可以是单速率的也可以是多速率的。多速率的 S-函数具有多个采样时间。

采样时间按右边的格式成对说明:[sample_time, offset_time]。以下是几个有效的采样时间。

[CONTINUOUS_SAMPLE_TIME, 0.0]

[CONTIMUOUS_SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]

[discrete_sample_time_period, offset]

[VARIABLE_SAMPLE_TIME, 0.0]

其中,字母小写的变量表示用户在设定时要用具体实数去替代它们,面字母大写的字符串是几个 Simulink 定义过的常量,它们的取值分别为:

CONTIMUOUS_SAMPLE_TIME = 0.0

FIXED_IN_MINOR_STEP_OFFSET = 1.0

VARIABLE_SAMPLE_TIME = -2.0

同样,用户可以设置采样时间从驱动模块继承,这时采样时间只能有一个采样时间对。

[INHERITED_SAMPLE_TIME, 0.0]

[INHERITED_SAMPLE_TIME, FIXED_IN_MINOR_STEP OFFSET]

其中,

INHERITED_SAMPLE_TIME = -1.0

下面的几个准则可以帮助用户说明采样时间。

(1)在最小积分步值会变化的连续 S-函数,采样时间应该设置为[CONTINUOUS_SAMPLE_TIME, 0.0]

(2)连续但在最小积分步值不变化的 S-函数,应该具有[CONTINUOUS_SAMPLE

_ TIME, FIXED _ IN _ MINOR _ STEP _ OFFSET]的采样时间。

(3)以一个固定速率变化的离散 S－函数应该具有离散时间对,[discrete _ sample _ time _ period, offset] 。这里

diserete _ sample _ period > 0.0

以及

0.0 < offset < discrete _ sample _ period

(4)以变速率执行的离散 S－函数,应该具有可变步长的离散采样时间: [VARIABLE _ SAMPLE _ TIME, 0.0]。

如果用户的 S－函数具有继承的采样时间,那么存在两种情况:

• 值随输入信号变化而变化,甚至在最小的积分步也是如此的 S－函数,应该设置采样时间为 [INHERITED _ SAMPLE _ TIME, 0.0] 。

• 值随输入信号变化而变化,但是在最小的积分步不变化的 S－函数,应该设置采样时间为 [INHERITED _ SAMPLE _ TIME, FIXED _ IN _ MINOR _ STEP _ OFFSET]。

10.2 建立 M 文件 S－函数

定义 S－函数的 M 文件要提供模型有关的信息,这些信息是仿真过程中为 Simulink 所必需的。仿真进行时,Simulink、ODE 解法器和 M 文件交互来完成特定的任务。这些任务包括定义初始条件和模块特性、计算微分、离散状态和输出。

10.2.1 如何使用模板

建立 M 文件 S－函数的推荐方法是使用 Simulink 提供的模板 M 文件——sfuntmpl.m,它的位置在 MATLAB 根目录下的 toolbox/Simulink/blocks 目录。读者可以用 MATLAB Edit 浏览这个模板文件。它里面的代码如下所示:

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
%SFUNTMPL General M - file S - function template
%  With M - file S - functions, you can define you own ordinary differential
%  equations (ODEs), discrete system equations, and/or just about
%  any type of algorithm to be used within a Simulink block diagram.
%  The general form of an M - File S - function syntax is:
%  [SYS,X0,STR,TS] = SFUNC(T,X,U,FLAG,P1,...,Pn)
%  What is returned by SFUNC at a given point in time, T, depends on the
%  value of the FLAG, the current state vector, X, and the current
%  input vector, U.
%  FLAG  RESULT            DESCRIPTION
%  ----- ------  ----------------------------------------------------
%  0     [SIZES,X0,STR,TS]  Initialization, return system sizes in SYS,
```

```
%                                  initial state in X0, state ordering strings
%                                  in STR, and sample times in TS.
%   1      DX                      Return continuous state derivatives in SYS.
%   2      DS                      Update discrete states SYS = X(n+1)
%   3      Y                       Return outputs in SYS.
%   4      TNEXT                   Return next time hit for variable step sample
%                                  time in SYS.
%   5                              Reserved for future (root finding).
%   9      []                      Termination, perform any cleanup SYS=[].
%
%   The state vectors, X and X0 consists of continuous states followed
%   by discrete states.
%   Optional parameters, P1,...,Pn can be provided to the S-function and
%   used during any FLAG operation.
%   When SFUNC is called with FLAG = 0, the following information
%   should be returned:
%      SYS(1) = Number of continuous states.
%      SYS(2) = Number of discrete states.
%      SYS(3) = Number of outputs.
%      SYS(4) = Number of inputs.
%               Any of the first four elements in SYS can be specified
%               as -1 indicating that they are dynamically sized. The
%               actual length for all other flags will be equal to the
%               length of the input, U.
%      SYS(5) = Reserved for root finding. Must be zero.
%      SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-function
%               has direct feedthrough if U is used during the FLAG=3
%               call. Setting this to 0 is akin to making a promise that
%               U will not be used during FLAG=3. If you break the
%               promise then unpredictable results will occur.
%      SYS(7) = Number of sample times. This is the number of rows in TS.
%      X0     = Initial state conditions or [] if no states.
%      STR    = State ordering strings which is generally specified as [].
%      TS     = An m-by-2 matrix containing the sample time
%             (period, offset) information. Where m = number of sample
%             times. The ordering of the sample times must be:
%             TS = [0     0,      : Continuous sample time.
%                   0     1,      : Continuous, but fixed in minor step
%                                   sample time.
```

```
%            PERIOD OFFSET Discrete sample time where
%                          PERIOD > 0 & OFFSET < PERIOD.
%            -2     0];    : Variable step discrete sample time
%                          where FLAG = 4 is used to get time of
%                          next hit.
%         There can be more than one sample time providing
%         they are ordered such that they are monotonically
%         increasing. Only the needed sample times should be
%         specified in TS. When specifying than one
%         sample time, you must check for sample hits explicitly by
%         seeing if
%             abs(round((T - OFFSET)/PERIOD) - (T - OFFSET)/PERIOD)
%         is within a specified tolerance, generally 1e-8. This
%         tolerance is dependent upon your model's sampling times
%         and simulation time.
%         You can also specify that the sample time of the S-function
%         is inherited from the driving block. For functions which
%         change during minor steps, this is done by
%         specifying SYS(7) = 1 and TS = [-1 0]. For functions which
%         are held during minor steps, this is done by specifying
%         SYS(7) = 1 and TS = [-1  1].

%     Copyright 1990-2000 The MathWorks, Inc.
%     $Revision: 1.16  $
% The following outlines the general structure of an S-function.
switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes;

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys = mdlDerivatives(t,x,u);
```

```
    %%%%%%%%%%
    % Update %
    %%%%%%%%%%
    case 2,
      sys = mdlUpdate(t,x,u);

    %%%%%%%%%%%
    % Outputs %
    %%%%%%%%%%%
    case 3,
      sys = mdlOutputs(t,x,u);

    %%%%%%%%%%%%%%%%%%%%%%%
    % GetTimeOfNextVarHit %
    %%%%%%%%%%%%%%%%%%%%%%%
    case 4,
      sys = mdlGetTimeOfNextVarHit(t,x,u);

    %%%%%%%%%%%%%
    % Terminate %
    %%%%%%%%%%%%%
    case 9,
      sys = mdlTerminate(t,x,u);

    %%%%%%%%%%%%%%%%%%%%
    % Unexpected flags %
    %%%%%%%%%%%%%%%%%%%%
    otherwise
      error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

% ==================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
% ==============================================================
```

```
%==============================================================
function [sys,x0,str,ts] = mdlInitializeSizes

% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
% Note that in this example, the values are hard coded. This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 0;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes= 1;     % at least one sample time is needed
sys = simsizes(sizes);

% initialize the initial conditions

x0 = [ ];

% str is always an empty matrix

str = [ ];

% initialize the array of sample times

ts = [0 0];

% end mdlInitializeSizes
%==============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%==============================================================

function sys = mdlDerivatives(t,x,u)
```

```
sys = [];

% end mdlDerivatives

%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================

function sys = mdlUpdate(t,x,u)

sys = [];

% end mdlUpdate

%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================

function sys = mdlOutputs(t,x,u)

sys = [];

% end mdlOutputs

%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is
% absolute time. Note that this function is only used when you specify a
% variable discrete - time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
```

```
%=================================================
===============================

function sys = mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;            % Example, set the next hit to be one second later.
sys = t + sampleTime;
% end mdlGetTimeOfNextVarHit

%=================================================
===============================
% mdlTerminate
% Perform any end of simulation tasks.
%=================================================
===============================

function sys = mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

下面就来简略地讲解模板的使用。

模板文件里 S - 函数的结构十分简单,它只为不同的 flag 的值指定要相应调用的 M 文件子函数,例如当 flag 值为 3 时,即模块处于计算输出这个仿真阶段时,相应调用的子函数为 sys = MdlOutputs (t, x, u);模板文件使用 switch 语句来完成这种指定,当然这种结构并不唯一,读者完全可以使用 if 语句来完成同样的功能。而且在实际运用时,可以根据实际需要来去掉某些值,因为并不是每个模块都要经过所有的仿真阶段。

读者要明白一点,模板文件只是 Simulink 为方便用户而提供的一种参考格式,并不是编写 S - 函数的语法要求。例如,读者完全可以把子函数取成别的名字,或者直接把代码写在主函数里。但是使用模板的一个好处,就是比较方便,而且条理清晰。

使用模板编写 S - 函数,用户只需要把 S - 函数名换成期望的函数名称,如果需要额外地输入参量,还需要在输入参数列表的后面增加这些参数,因为前面的四个参数是 Simulink 调用 S - 函数时自动传入的。面对于输出参数,最好不要做任何修改。主函数基本上就可以,不要读者再进行别的修改了。接下去的工作就是根据所编 S - 函数要完成的任务,用相应的代码去替代模板里各个子函数的代码,因为模板里最初的代码实际上什么也没有做。

下面就来介绍完成各个仿真阶段的子函数的编写方法。

首先,解决一个容易让 S - 函数新手困惑的问题。无论是在哪个仿真阶段,相应的 S - 函数方法的返回变量都是 sys,就像模板 M 文件显示的一样,计算得到的输出、更新好

的离散状态都是用sys变量返回。要弄清楚这个问题，还是要回到Simulink如何调用S-函数上来。前面讲过，Simulink在每个仿真步的仿真循环中的每个仿真阶段都会对S-函数进行调用。在调用时，Simulink不但根据所处的仿真阶段为flag传入不同的值，而且还会为sys这个返回变量指定不同的角色，也就是说尽管是相同的sys变量，但是在不同的仿真阶段其意义却不相同，这种变化由Simulink自动完成。

表10-2列出了M文件S-函数可用的S-函数方法。

表10-2　M文件S-函数方法

S-函数方法	说　明
mdlInitializesizes	定义S-函数模块的基本特性，包括采样时间，连续或者离散状态的初始条件和sizes数组
MdlDerivatives	计算连续状态变量的微分
MdlUpdate	更新离散状态，采样时间和主时间步的要求
MdlOutputs	计算S-函数的输出
MdlGetTimeOfNextVarHit	计算下一个采样点的绝对时间，这个方法仅仅是在用户在MdlInitializesSizes说明了一个可变的离散采样时间
mdlTerminate	实现仿真任务必需的结束

概括地说，建立S-函数可以被当成两个分离的任务：

(1)初始化模块特性包括输入输出信号的宽度，离散连续状态的初始条件和采样时间。

(2)将算法放到合适的S-函数方法中去。

10.2.2　定义S-函数的初始信息

为了让Simulink识别出一个M文件S-函数，用户必须在S-函数里提供有关S-函数的说明信息，包括采样时间、连续或者离散状态个数等初始条件。这一部分主要是在mdlInitializesizes方法里完成，请读者对照模板文件，来学习它的代码编写。在所有的M文件S-函数方法编写中，这个方法的语法约束应该是最多的。

读者可以看到在模板文件的MdlInitializesizes方法中，首先出现的语句是

```
sizes = simsizes;
```

这个语句返回一个没有经过初始化的sizes结构，sizes结构是S-函数信息的载体，它内部的字段，可以在MATLAB目录窗口查询：

```
> > sizes = simsizes;
> > sizes
sizes =
          NumConTStates:    0
          NumDiscStates:    0
          NumOutputs:       0
          NumInputs:        0
          DirFeedthrotgh:   0
```

NumSampleTime:　　0

上面的结果反映了 sizes 是一个具有 6 个字段的结构，它里面的字段的值在开始时被置为零。往 sizes 结构载入 S－函数的说明信息，就是重置这些字段的值。表 10－3 列出了这些字段的意义。

表 10－3　sizes 各个字段的意义

字段名	说　明
Sizes _ NumContStates	连续状态的个数(状态向量连续部分的宽度)
Sizes _ NumDiscStates	离散状态的个数(状态向量离散部分的宽度)
Sizes _ numOutputs	输出的数目个数(输出向量的宽度)
Sizes _ NumInputs	输入的数目个数(输入向量的宽度)
Sizes _ DirFeedthrough	有无直接馈入
Sizes _ NumSampleTimes	采样时间的个数

表中的这些字段的意义非常明确，在建立S－函数时，我们可以很方便地根据 S－函数的情况来对这些字段赋值。这里还是以前面曾提到过的S－函数 timestwo 来举例。它的 mdlInitializesizes 的代码如下：

```
% ========================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
% ========================================================================

function [sys,x0,str,ts] = mdlInitializeSizes()

sizes = simsizes;
sizes.NumContStates   = 0;
sizes.NumDiscStates   = 0;
sizes.NumOutputs      = -1;   % dynamically sized
sizes.NumInputs       = -1;   % dynamically sized
sizes.DirFeedthrough  = 1;    % has direct feedthrough
sizes.NumSampleTimes= 1;

sys = simsizes(sizes);
str = [];
x0 = [];
ts = [-1 0];                  % inherited sample time
```

% end mdlInitializeSizes

这个 S - 函数的作用是把输入信号乘以 2 倍，只需直接对输入信号操作即可，而不需要任何的状态变量来保存系统的一些内部状况(关于状态的理解，本章后面会用一个例子来说明)。因此，在说明 S - 函数信息时，连续状态数和离散状态数都被赋值为 0。而对于函数的输入向量和输出向量的宽度，由于是受驱动模块控制的，所以属于动态可变，为此，要把 NumOutputs 和 NumInputs 赋值为 - 1 来说明它们是动态可变的。如果输入输出的宽度是固定的，那么就要给这两个字段赋以实际的宽度值。sizes 结构的前面四个字段其实都是可以赋值为 - 1，来表示它们的值是动态可变的。

要注意 DirFeedthrough 是一个布尔变量，它的取值只有 0 和 1 两种，0 表示没有直接馈入，1 表示有直接馈入。如果 DirFeedthrough 的值被赋为 0，那么读者在编写 MdlOutputs 时就要确保子函数的代码里不出现输入变量 u，否则会出现无法预计的结果。

NumSampleTimes 表示采样时间的个数，也就是 ts 变量的行数，与用户对 ts 的定义有关。

对 sizes 赋值完毕后，再调用 simsizes，这时要传入定义好的 sizes 结构作为参数，并把结果返回给 sys 变量，相应的语句是：

```
sys = simsizes (sizes ) ;
```

其实，simsizes (sizes)只是把 sizes 结构组合成一个长度为 6 的向量。下面的命令证明了这个事实：

```
> >sizes.NumConstStates = 1;
> >sys = simsizes ( sizes);
sys =
           1    0    0    0    0    0    0
```

所以，完全可以不需要前面那么一大段的步骤，而直接对 sys 进行赋值，例如 timestwo 的 sys 变量就直接可以写为：

```
> >sys=[0, 0, -1, -1, 1, 1];
```

但是，提醒用户，赋值时一定要小心，避免出现输入错误。

在这个小节的最后，我们再来强调一下如何确定 S - 函数的输入、输出个数。例如，要编写一个描述将两个输入信号相加的 S - 函数。封装好的 S - 函数模块在外形上就应该有两个输入端口，而每个端口又有多个元素。前面曾讲过，S - 函数会忽略端口。事实上，S - 函数的输入个数被设为所有端口的元素个数的总和，而不是端口的个数，也就是说所有的输入端口都会并到一个输入向量中。但是模块是通过两个端口来输入，于是在使用 S - 函数时，要用一个 Mux 模块将两个信号合并成一个信号再连到 S - function 模块。

10.2.3 输入和输出参量说明

S - 函数前面的四个输入参量，必须是 t, x, u 和 flag，并且次序不能变动。它们的意义分别是：

(1)t 代表当前的仿真时间，这个输入参数通常用于决定下一个采样时刻，或者在多个采样速率系统中，来区分不同的采样时刻点，并据此进行不同的处理。

(2)x 表示状态向量，这个参数是必需的，甚至在系统中不存在状态时也是如此的。

状态有很灵活的运用，本章后面会有一个例子来说明。

(3)u 表示输入向量。

(4)flag 是一个控制在每一个仿真阶段调用哪种 S－函数方法的参数，由 Simulink 在调用时自动取值。

Simulink 也需要四个返回参数——sys、x0、str 和 ts，它们的排列顺序也必须是模板所指定的顺序。这些参数的意义为：

(1)sys 是一个通用的返回参量，它所返回值的意义取决于 flag 的值，例如，当 flag 的值是 3 时，sys 包含的是 S－函数的输出。

(2)x0 是初始的状态值(没有状态时，就是一个空矩阵[])，这个返回参量只在 flag 值为 0 时才有效，其他时候都会被忽略。

(3)str 这个参量没有什么意义，是 MathWorks 公司为将来的应用保留的，M 文件S－函数必须把它设为空矩阵[]。

(4)ts 是一个 m×2 的矩阵，它的两列分别表示采样时间间隔和偏移。关于不同的采样时间，如何进行设置，在前面已经提到过，这里就不再详述了。ts 可以被赋为一个多行的矩阵，这时矩阵所有行的采样时间必须按单调递增的顺序排列。例如，读者想让 S－函数在[0, 0.1, 0.25, 0.75, 1.0, 1.1, 1.25,…]执行，可以为 ts 设置 2×2 的矩阵，即

```
ts = [0.25, 0; 1.0, 0.1];  %多采样速率的设置
```

10.2.4　M文件S－函数的几个示例

下面分别就连续状态系统、离散状态系统、混合系统和变步长系统各自举一个 S－函数例子，来加深对用 M 文件编程的理解。最后，还举了一个移位寄存器序列的例子来加深状态向量的理解。

1. 连续状态 S－函数示例

这个例程定义了一个连续状态空间系统，它的状态方程为

$$x = Ax + Bu$$
$$y = Cx + Du$$

从这个例子，读者应该强化的几个概念是：如何在初始化方法里，定义 S－函数的有关信息，如何根据所描述的系统来确定连续和离散状态数，输入和输出信号数。在这个例子中不存在离散状态，所以离散状态数就被设置为 0，而连续状态向量(微分方程所对应的)的宽度为 2，所以连续状态数就是 2。而对于输入和输出，它们的信号宽度都是 2，所以相应的描述参数也被设为 2。这里要记住一点，输入信号数 sizes.NumOutputs 和输出信号数 sizes.NumInputs，指的是信号的宽度，而不是模块里的端口个数，也就是说，它们是模块里所有输入端口(输出端口)的信号宽度之和。

还应该记住的一点是，S－函数方法 MdlDerivatives 返回的是连续的微分值，Simulink 会把它作为微分值来处理，从而计算连续状态的积分。

在例子中，状态空间方程中的参数 A、B、C、D 都是作为固定值在程序中指定的，其实，完全可以把它们作为参数由使用者在外面设定，就像 Simulink 模块 State Space 模块所表现的那样。这时，就要根据这些参数在 MdlInitializesSizes 方法中动态地设定各个函数的描述信息值。实现的方法很容易想出，不清楚的读者可以参考本书的最后一个例程。

下面是这个例程的代码(例 1 至例 4 的 M 文件 S - 函数代码可以在 simulink/blocks 目录里找到,在这个目录里,还提供了其他的一些示例):

```
function [sys,x0,str,ts] = csfunc(t,x,u,flag)
%CSFUNC An example M-file S-function for defining a continuous system.
%      Example M-file S-function implementing continuous equations:
%         x' = Ax + Bu
%         y = Cx + Du
%      See sfuntmpl.m for a general S-function template.
%      See also SFUNTMPL.

%      Copyright 1990-2000 The MathWorks, Inc.
%      $Revision: 1.7 $

A=[-0.09    -0.01
    1          0];

B=[1    -7
   0    -2];

C=[0    2
   1    -5];

D=[-3   0
    1   0];
switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u,A,B,C,D);

  %%%%%%%%%%%
```

```
% Outputs %
%%%%%%%%%%%
case 3,
  sys = mdlOutputs(t,x,u,A,B,C,D);

%%%%%%%%%%%%%%%%%%%
% Unhandled flags %
%%%%%%%%%%%%%%%%%%%
case {2, 4, 9},
  sys = [];

%%%%%%%%%%%%%%%%%%%%
% Unexpected flags %
%%%%%%%%%%%%%%%%%%%%
otherwise
  error(['Unhandled flag = ',num2str(flag)]);

end
% end csfunc
%====================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%====================================================================
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)

sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0 = zeros(2,1);
str = [];
ts = [0 0];
```

```
% end mdlInitializeSizes
%=====================================
==============================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=====================================
==============================
%
function sys = mdlDerivatives(t,x,u,A,B,C,D)

sys = A * x + B * u;

% end mdlDerivatives
%=====================================
==============================
% mdlOutputs
% Return the block outputs.
%=====================================
==============================
function sys = mdlOutputs(t,x,u,A,B,C,D)

sys = C * x + D * u;

% end mdlOutputs
```

2、离散状态 S－函数示例

这个例程则定义了一个离散线性系统，它实现的离散方程是：

$$x(n+1) = Ax(n) + Cu(n)$$

$$y(n) = Cx(n) + Du(n)$$

编写离散状态 S－函数，需要调用对状态进行更新的 S－函数方法 mdlUpdates。其余的地方和连续状态 S－函数的编写区别不大。整个文件的代码如下：

```
function [sys,x0,str,ts] = dsfunc(t,x,u,flag)
%DSFUNC An example M-file S-function for defining a discrete system.
%      Example M-file S-function implementing discrete equations:
%         x(n+1) = Ax(n) + Bu(n)
%         y(n) = Cx(n) + Du(n)
%      See sfuntmpl.m for a general S-function template.
%      See also SFUNTMPL.
%      Copyright 1990-2000 The MathWorks, Inc.
```

```
%      $ Revision: 1.16 $
% Generate a discrete linear system:
A=[-1.3839     -0.5097
     1.0000          0];

B=[-2.5559           0
     0         4.2382];

C=[       0     2.0761
          0     7.7891];

D=[     -0.8141     -2.9334
         1.2426           0];

switch flag,
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D);
  case 2,
    sys = mdlUpdate(t,x,u,A,B,C,D);
  case 3,
    sys = mdlOutputs(t,x,u,A,C,D);
  case {1,4,9} % Unused flags
    sys = []; % do nothing
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
%end dsfunc
%=====================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=====================================================================
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 2;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 2;
```

```
sizes.DirFeedthrough  = 1;      当矩阵 D 不为空
sizes.NumSampleTimes= 1;
sys  = simsizes(sizes);
x0   = ones(2,1); % 初始化离散状态
str  = [];
ts   = [1 0]; %采样时间:[period, offset]
% end mdlInitializeSizes

%===================================
==============================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%===================================
==============================
function sys = mdlUpdate(t,x,u,A,B,C,D)
sys = A * x + B * u;
%end mdlUpdate

%===================================
==============================
% mdlOutputs
% Return the output vector for the S-function
%===================================
==============================
function sys = mdlOutputs(t,x,u,A,C,D)

sys = C * x + D * u;

%end mdlOutputs
```

3. 混合系统 S－函数

这个 M 文件定义了一个描述混合系统(连续状态和离散状态的结合)的 S－函数。这个 S－函数所描述的系统等价于一个积分器后接一个离散单位时间延迟的混合系统(如图 10－7 所示)。

在混合系统中,与离散状态和连续状态有关的 S－函数方法,都有可能执行,但是显然各自的执行时间是不一样的。如积分器在每一个微时间步都会执行,而单位时间延迟只在离散采样时刻到达时执行。

下面就简要地描述一下这个例程的编写要点。首先,例程在 mdlInitializesSizes 方法下定义了两个采样时间:连续采样时间和由离散采样周期决定的离散采样周期。于是在

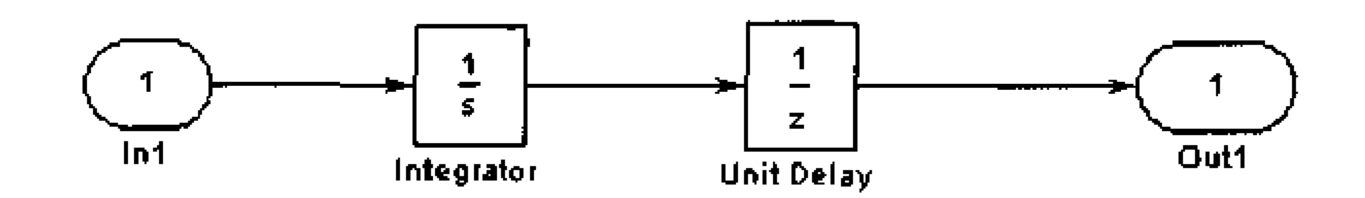


图 10－7 混合系统

这两个采样时间决定的采样点，Simulink 都会调用 S－函数文件，并依次用 flag 来指定要调用仿真循环中的哪个方法。

我们知道，尽管连续状态的微分计算是在每一个微时间步都要进行的，但离散状态的更新以及系统的输出的产生，只是在离散采样点到达时才完成。但这一点 Simulink 是无法区分什么时候该更新什么时候不该更新。因为 Simulink 从初始化获得的信息，既有离散状态又有连续状态。据此它会决定改函数的仿真循环应该包含连续状态的微分计算和离散状态更新这两个方法，而没有足够的信息来判断当前时刻是不是离散采样点。

这时常用的一个技巧是，在 MdlUpdate 和 MdlOutputs 中由程序显示判断当前时刻是不是离散采样点。完成它的语句是：

```
if abs (round ( (t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e - 8
    sys = x (1) ;         %表示是离散采样点，就把离散状态更新为当前的连续状态
else
    sys = [] ;            %否则离散状态就不改变
End
```

MdlOutputs 方法中的判断语句也是极为类似的。整个文件的代码如下：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
%MIXEDM An example integrator followed by unit delay M - file S - function
%     Example M - file S - function implementing a hybrid system consisting
%     of a continuous integrator (1/s) in series with a unit delay (1/z).
%     See sfuntmpl.m for a general S - function template.
%     See also SFUNTMPL.
%     Copyright 1990 - 2000 The MathWorks, Inc.
%     $Revision: 1.26 $

% Sampling period and offset for unit delay.
dperiod = 1;
doffset = 0;

switch flag
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(dperiod,doffset);
```

```
    case 1
      sys = mdlDerivatives(t,x,u);
    case 2,
      sys = mdlUpdate(t,x,u,dperiod,doffset);
    case 3
      sys = mdlOutputs(t,x,u,doffset,dperiod);
    case {4,9}
      sys = [];

    otherwise
      error(['unhandled flag = ',num2str(flag)]);

  end
  % end mixedm

  % = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =
  % mdlInitializeSizes
  % Return the sizes, initial conditions, and sample times for the S - function.
  % = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =

  function [sys,x0,str,ts] = mdlInitializeSizes(dperiod,doffset)

  sizes = simsizes;
  sizes.NumContStates  = 1;
  sizes.NumDiscStates  = 1;
  sizes.NumOutputs     = 1;
  sizes.NumInputs      = 1;
  sizes.DirFeedthrough = 0;
  sizes.NumSampleTimes = 2;
  sys = simsizes(sizes);
  x0  = ones(2,1);
  str = [];
  ts  = [0 0; dperiod doffset];
  % end mdlInitializeSizes

  % = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
% mdlDerivatives
% Compute derivatives for continuous states.
%===================================================================

function sys = mdlDerivatives(t,x,u)
sys = u;
% end mdlDerivatives

%===================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%===================================================================

function sys = mdlUpdate(t,x,u,dperiod,doffset)

% next discrete state is output of the integrator
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);
%是一个离散采样点,更新离散状态值为连续状态值
else
  sys = [];
%不是一个离散采样点,返回一个空矩阵,表示没有变化
end
% end mdlUpdate

%======================================================================% mdlOutputs
% Return the output vector for the S-function
%===================================================================

function sys = mdlOutputs(t,x,u,doffset,dperiod)
% Return output of the unit delay if we have a
% sample hit within a tolerance of 1e-8. If we
% don't have a sample hit then return [] indicating
```

```
% that the output shouldn't change.
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(2);
else
  sys = [];
end

% end mdlOutputs
```

4. 可变步长仿真系统示例

这是一个可变仿真步长系统的 S－函数示例，也就是说仿真的步长在运行过程中可以动态变化。就像本例中，S－函数实现对输入的第一个元素的延迟输出，而延迟的时间由输入的第 2 个元素决定。在 S－函数中，实现这一点主要使用 MdlGetTimeOfNexrVarHit 方法，它的作用是决定下一个采样点的时间，对应的 flag 参数是 4 。要使 Simulink 在仿真中调用这个方法，需要先在 MdlInitializesSizes 方法里说明采样时间是变采样时间，设置方法如下：

```
ts = [-2, 0];
```

在初始化模块信息时，还要注意直接馈入的设置。因为在 S－函数中采样时间的计算和输入直接有关，所以这个 S－函数是具有直接馈入的。

初始化信息之后，就可以在 mdlGetTimeOfNextVarHit 方法里计算出下一个采样时间。其代码是

```
sys = t + u(2)
```

注意，返回值同样是传给 sys 变量，而不要犯 t = t + u(2)的错误。

下面是这个例子的源代码：

```
function [sys,x0,str,ts] = vsfunc(t,x,u,flag)
%VSFUNC Variable step S-function example.
%      This example S-function illustrates how to create a variable step
%      block in Simulink. This block implements a variable step delay
%      in which the first input is delayed by an amount of time determined
%      by the second input:
%      dt = u(2)
%      y(t+dt) = u(t)
%      See also SFUNTMPL, CSFUNC, DSFUNC.
%      Copyright 1990-2000 The MathWorks, Inc.
%      $Revision: 1.8 $
% The following outlines the general structure of an S-function.

switch flag,

  %%%%%%%%%%%%%%%%%%
```

```
% Initialization %
%%%%%%%%%%%%%%%%%%
case 0,
  [sys,x0,str,ts] = mdlInitializeSizes;

%%%%%%%%%%
% Update %
%%%%%%%%%%
case 2,
  sys = mdlUpdate(t,x,u);

%%%%%%%%%%%
% Outputs %
%%%%%%%%%%%
case 3,
  sys = mdlOutputs(t,x,u);

%%%%%%%%%%%%%%%%%%%%%%%
% GetTimeOfNextVarHit %
%%%%%%%%%%%%%%%%%%%%%%%
case 4,
  sys = mdlGetTimeOfNextVarHit(t,x,u);

%%%%%%%%%%%%%
% Terminate %
%%%%%%%%%%%%%
case 9,
  sys = mdlTerminate(t,x,u);

%%%%%%%%%%%%%%%%%%%
% Unhandled flags %
%%%%%%%%%%%%%%%%%%%
case 1,
  sys = [];

%%%%%%%%%%%%%%%%%%%%
% Unexpected flags %
%%%%%%%%%%%%%%%%%%%%
otherwise
```

```
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl
% ==============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
% ==============================================================================

function [sys,x0,str,ts] = mdlInitializeSizes

% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 1;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;      % at least one sample time is needed

sys = simsizes(sizes);

% initialize the initial conditions

x0 = [0];

% str is always an empty matrix

str = [];

% initialize the array of sample times

ts = [-2 0];                   % variable sample time
```

```
% end mdlInitializeSizes

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =

function sys = mdlUpdate(t,x,u)

sys = u(1);

% end mdlUpdate

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =
% mdlOutputs
% Return the block outputs.
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =

function sys = mdlOutputs(t,x,u)

sys = x(1);

% end mdlOutputs

% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is
% absolute time.
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = =

function sys = mdlGetTimeOfNextVarHit(t,x,u)
```

```
sys = t + u(2);

% end mdlGetTimeOfNextVarHit

%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================

function sys = mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

5. 最大长度线性移位寄存器序列的生成

这个例子主要是为了加深对状态的理解。前面的那些例子中的状态概念有浓厚的信号与系统理论背景。但是从编程的角度来讲，状态也可以理解为一个全局变量，函数编写者可以用状态（离散状态）来保留一些全局信息。

本例描述的一个线性移位寄存器序列发生器系统，如图 10－8 所示。这里不讨论这个系统的理论背景，感兴趣的读者可以参阅有关的书籍。

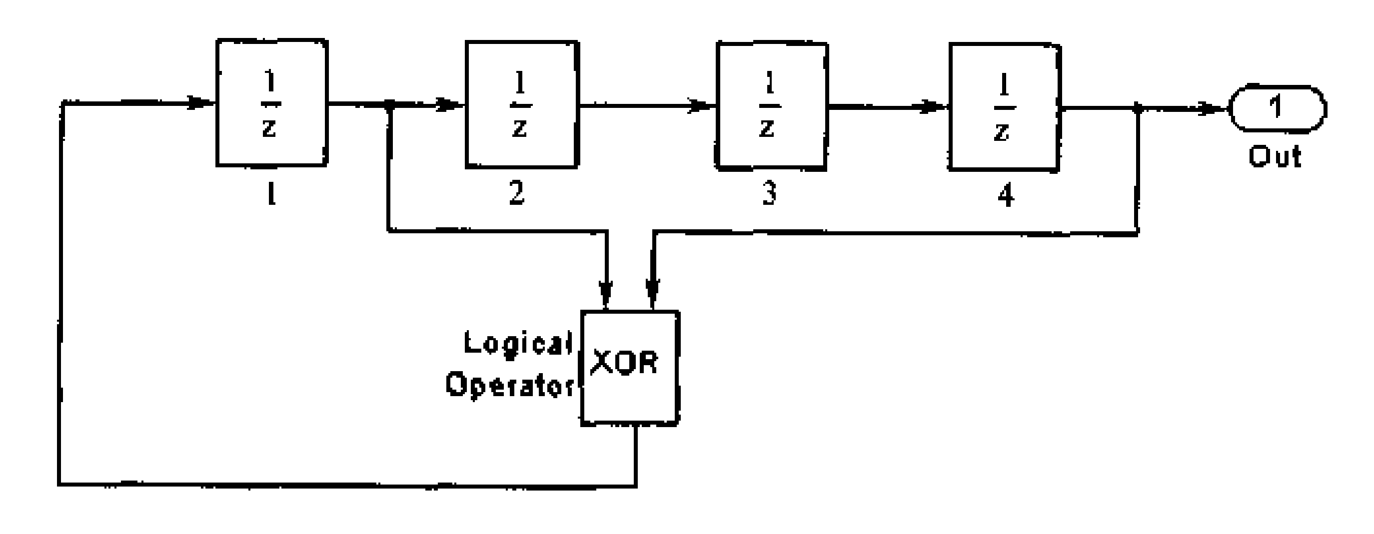


图 10－8 一个 4 级移位寄存器序列

从图中，读者可以看出，系统由一系列单位时延模块串接而成，模块的个数也称为级数。而这些模块中某些输出被连到一个异或逻辑运算模块进行异或（模 2 加），得出的结果再输出到第一个时延的输入，最后一个时延模块的输出作为系统的输出。在每一个时延模块的初始状态设为 0 或者 1 的情况下，执行模型，即可以得到一个 0、1 二进制序列。

至于每个模块是否连接到异或模块进行运算的状态，可以用一个向量 poly 来表示。对于 m 级系统，向量 poly = [c1; c2; c3; c4; …cm]，它的元素要为 1。如果第 i 个元素为

0 表示第 I 个单元不要连接到异或模块，为 1 则表示要连接。例如图 10－8 延时的系统对应的向量为[1; 0; 0; 1]。

下面的 S－函数将 poly、单位延迟时间(periof)和初始状态向量(ini _ st)作为额外参数，这样就可以让使用者在外面来设置这些参数，生成不同级数，不同连接情况，不同周期和不同初始状态的移位寄存器序列。

改函数，读者要注意如何在初始化时，根据传入的参数来动态地变化 S－函数的有关信息，这里是离散状态数。

另外一点，本例是通过用一个长度和级数相同的离散状态向量来表示串接的单位延迟模块。在每个采样时刻，通过将状态向量左移位就可以表示单位延迟的串接。S－函数的代码如下所示。

```
function [sys,x0,str,ts] = pnsequence(t, x, u, flag,poly,period,ini _ st)
switch flag

  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(poly, period,ini _ st);

  case 2,
    sys = mdlUpdate(t,x,u,poly);

  case 3,
    sys = mdlOutputs(t,x,u);

  case.{1,4,9},
    sys = []
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

% end sfuntmpl

% mdlInitializesSizes

function [sys,x0,str,ts] = mdlInitializeSizes(poly, period, ini _ st)

  n _ dis = length(poly);
  sizes = simsizes;
  sizes.NumContStates = 0;
  sizes.NumdsicStates = n _ dis;
  sizes.NumOutputs = 1;
```

```
    sizes.NumInputs  =0;
    sizes.DirFeedthrough  =0;
    sizes.NumSampleTimes  =  1;    % at least one sample time is needed
    sys  =  simsizes(sizes);
    x0=[ini_st, 0];
    str=[];
    ts=[period 0];
    % end mdlInitializeSizes

    %mdlUpdate

function sys=mdlOutputs(t,x,u,poly)
      feg=x;
      n_x=length(x);
      temp=mod(poly * reg(1:n_x-1), 2);  % 完成异或
      reg=[ret(2:n_x) ;temp];            % 进行左移位
      sys=reg;
      % end ndlUpdate

      % mdlOutputs
      functiong sys mdlOutputs (t,x,u)
      sys=x(1);

      % end mdlOutputs
```

10.3 C MEX S-函数

M 文件 S-函数的优点是编写简单,但是它会影响仿真的运行速度,而且包含 M 文件 S-函数的模型无法生成实时代码,无法利用 RTW 提供的许多强大功能。所以,如果想利用 S-函数高效地对 Simulink 进行扩展,那么就必须掌握 C MEX S-函数的编写方法。这里只对其中作一个简单的介绍,更详细的信息,请参阅 Simulink 的帮助文档。

10.3.1 介绍

同 M 文件 S-函数一样,在仿真时,一个 S-函数模块必须提供给 Simulink 有关的模型信息。当仿真进行时,Simulink、ODE 解法器和 MEX 文件交互地实现指定的任务。这些任务包括定义初始条件和模块特性、计算微分、离散状态和输出,它们的定义和 M 文件 S-函数的一样。

确切地说,C MEX S-函数有着和 M 文件 S-函数相同的结构,它能够实现 M 文件 S-函数能实现的功能。而且,C MEX S-函数比 M 文件 S-函数为用户提供更多的功

能。与 M 文件 S－函数类似，Simulink 同样提供了模板文件 sfuntmpl.c 以及它的复杂版本 sfuntmpl.doc(都在 Simulink/src 目录下)，来方便读者编写 C MEX S－函数。

C MEX S－函数源文件的通用格式如下所示：

```
# define S_FUNCTION_NAME your_sfunction_name_here
% 在这里定义 S－函数名
# define S_FUNCTION_LEVEL 2
# include "simstruc.h"
static void mdlInitializeSizes (SimStruct * S)
{
}
<additional S-function routines/code>
static void mdlTerminate(SimStruct * S)
{
}
# dfdef MATLAB_MEX_FILE /*  这个文件是编译为一个 MEX 文件吗？ ×、
# include "simulink.c" /*    MEX-file interface mechanism */
# else
# include "cg_sfun.h" /*      代码生成注册函数×、
# endif
```

mdlInitializeSizes 是 Simulink 在和 S－函数交互时首先调用的方法。在一个 S－函数源文件内，还有其他的几个 mdl＊方法。所有的 S－函数源文件必须和这个格式一致，也就是说不同的 S－函数文件内的方法可以使用相同的名称(如 mdlInitializeSizes)，因为不同的 S－函数文件是通过 S－函数的名称来区别的。在 Simulink 调用 mdlInitializeSizes 后，通过其他的几种方法(都以 mdl 开头)来和 S－函数实现交互，在仿真结束时，mdlTerminate 就被调用。

和 M 文件 S－函数不同，Simulink 不是通过显式的 flag 参数来指定调用何种 C MEX S－函数方法。这是因为 Simulink 在交互作用时，会自动地在合适的时候调用每个 S－函数方法。同样，C MEX S－函数中的某些方法在 M 文件 S－函数中，可以找到对应项。

Simulink 保存 S－函数的有关信息在一个名为 SimuStruct 的数据结构中。宏语句 # include"simtru.ch"定义了 SimStruct，它使用户的 MEX－文件能在 SimStruct 中赋值和从中取值。

10.3.2　编写基本的 C MEX S－函数

这一节讨论如何建立基本的 C MEX S－函数，这里的基本是指仅仅包含具有规定的 S－函数方法的 S－函数。然而，这些方法的内容可以任意的复杂。用户可以随意地把任何逻辑代码放入 S－函数方法，只要它符合规定的格式。这里，就以前面一节提过的 timetwo 的 C MEX S－函数版本 doubleinput 作为示例。

学习过 M 文件 S－函数编写的读者，知道 S－函数通过 Simulink 提供的 S－Function 模块和 Simulink 模型合并起来使用。那么 C MEX S－函数呢？它同样也是通过这个模

块。请读者看图 10－9 所示的模型。模型中，doubleinput S－函数的作用是将正弦波的幅度加倍，并且显示在 scope 上。

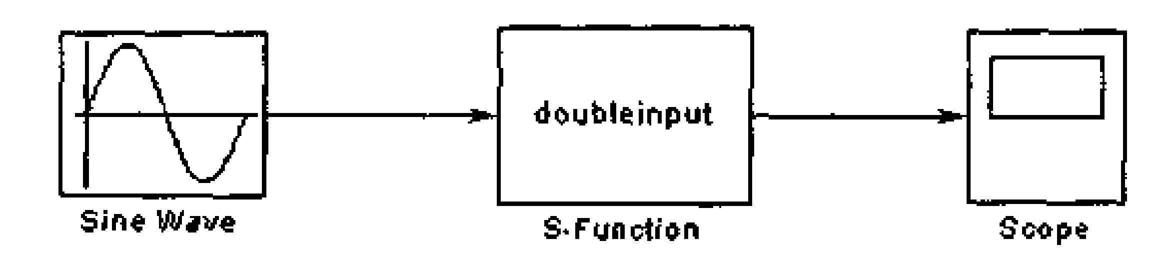


图 10－9 将函数合并到 Simulink 模型中

S－Function 模块对话框的设置和 M 文件 S－函数一样，name 参数的值为 doubleinput，parameters 参数编辑框则空着不用填写。但是 C MEX S－函数和 M 文件 S－函数不同的一点是，M 文件 S－函数编好之后就可以使用；但是 C MEX S－函数则不然，在 S－函数的源文件写好之后，还必须使用 mex 目录将其编译为 MEX 可执行文件。例如，在命令行输入

```
>>mex doubleinput.c
```

这命令将使 MATLAB 编译和连接 doubleinput.c 文件，将生成一个供 Simulink 使用的动态装载的可执行文件。这个可执行文件就成为 MEX S－函数，其中，MEX 代表"MATLAB Executable"。MEX 文件的扩展名因平台而异，在微软视窗环境下，其扩展名就是.dll。

mex 是 MATLAB 的编译命令，它通常把编译好的 MEX 文件保存在当前工作目录中，所以读者在使用改目录之前，最好是在路径浏览器里把当前目录设置为读者自己的工作目录。至于 mex 命令的详细说明，请读者自行查阅帮助。

图 10－10 显示了 doubleinput.c 所包含的 S－函数方法。

表 10－4 则列出了这些 S－函数方法的描述。它们的功能和 M 文件 S－函数中的对应方法大致相同。

表 10－4 S－函数方法描述

S－函数方法	说　明
MdlInitializesSizes	当 Simulink 开始处理模型并决定输入和输出端口的数目时，就调用这个方法。在仿真开始时 Simulink 同样会调用这个方法，以获得函数的信息，如端口的尺寸以及状态数(和 M 文件 S－函数类似)
MdlInitializeSampleTimes	Simulink 调用这个方法来设置 S－函数采样时间。doubleinput 有一个单一的继承采样时间，SAMPLE_TIME_INHERITED
MdlOutputs	计算输出。在本例中，将输入信号乘以 2，再放入输出信号中。这个方法在仿真循环中输出需要更新的每个时间步被调用，这里就是每个仿真时间步
MdlTerminate	实现在仿真结束时的操作。doubleipnut 因为没有任何操作要进行，所以为零

doubleinput.c 代码如下：

```
# define S_FUNCTION_NAME doubleinput
# define S_FUNCTION_LIVEL 2
```

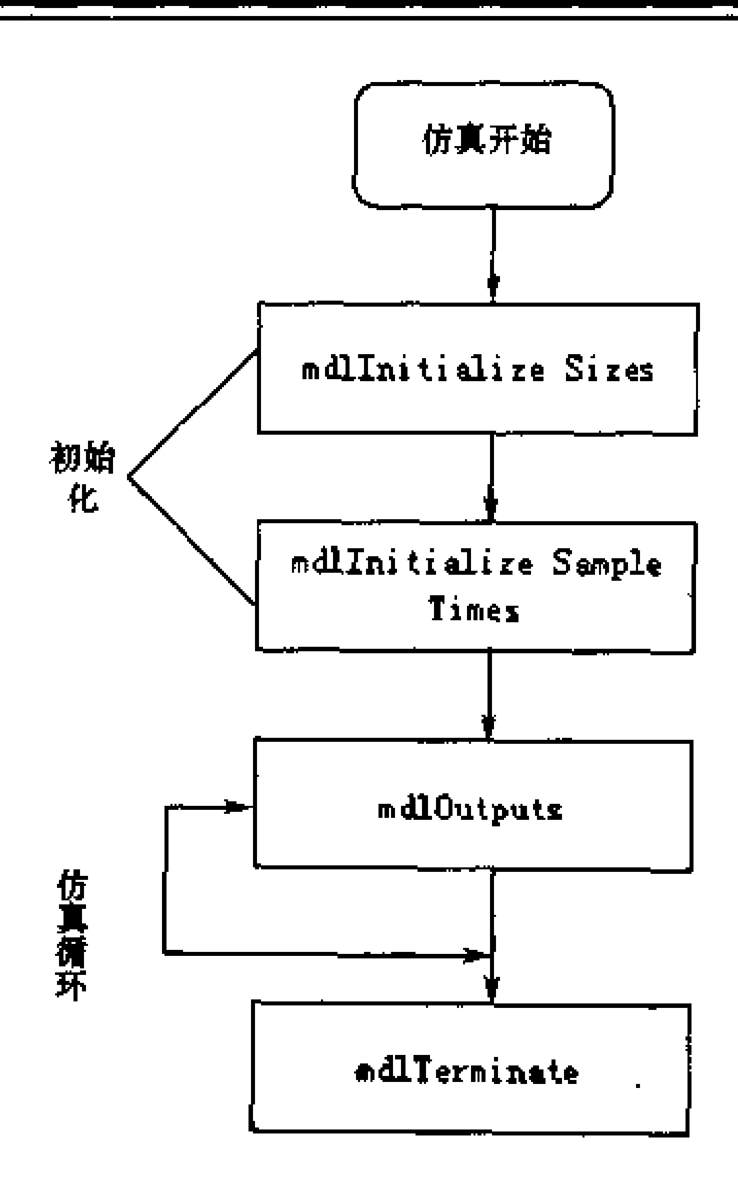


图 10－10　doubleinputs 中 S－函数方法

```
# include "simstruc.h"
/* Function: mdlInitializeSizes
 * Abstrace:
 * Serup sezes of the varionts vectors.
 */
static void mdlInitializeSizes (SimStruct * S)
{
  ssSetNumSFcnParams (S,0) ;
  if ( ssGetNumSFcnParams (S) ! = ssGetSFcnParamsCount (S)) {
    return;  /* Parameter mismatch will be reported by Simulink */
  }
  if (! ssSetNumInputPorts (S, 1)) return;
  ssSetInputPortWidth (S, 0, DYNAMICALLY _ SIZED);
  ssSetInputPortDirectFeedThrough (S, 0, 1) ;
  if (! ssSetNumOutputPorts (S,1))  return;
  ssSetOutputPortWidth (S, 0, DYNAMICALLY _ SIZED) ;
  ssSetNumSampleTimes (S, 1) ;
  /* Take care when specifying exception free code - see sfuntmpl.doc */
  ssSetOptions ( S, SS _ OPTION _ EXCEPTION _ FREE _ CODE) ;
}
```

```
/* Function: mdlInitializesSampleTimes
 * Abstract:
 *   Specify that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes (SimStruct *S)
{
ssSetSampleTime (S, 0, INHERITED_SAMPLE_TIME) ;
    ssSetOffsetTime (S, 0, 0.0) ;
}
/* Function: mdlOutputs
 * Abstract:
 * y = 2*u
 */
static void mdlOutputs (SimStruct *S, int_T tid)
{
    int_T i ;
    InputRealPtrsType uPtrs = ssGEtInputPortRealSignalPtrs (S,0) ;
    Real_T *y = ssGetputPortRealSignal (S,0);
    Int_T width = ssGetOutputPortWidth (S,0);
    For (i=0; i<width;i++) {
        *y++ =2.0 * (*uPtrs[i]) ;
        }
}
/* Function: mdlTerminate
 * Abstract:
 * No termination needed, but we are required to have this routine.
 */
static void mdlTerminate (SimStruct *S)
{
}
# ifdef MATLAB_MEX_FILE / * Is this file being compliled as a MEX - file?
*/
# include "Simulink.c"          /* MEX-file interface mechanism */
# else
# include"eg_sfun.h'            /* Code generation registration function */
# endif
```

读者写这个源文件时，无需自己来全部输入，比较好的建议是使用 Simulink 提供的模板文件 sfuntmpl.c。它的使用方法和 M 文件 S-函数模板使用是一样的，读者只需把

各个 S－函数方法的代码放到相应的位置，这样那些比较固定的语句，就不需要读者自己输入了。对于这个例子，还有一个更简单的方法。其实这里的 doubleinput.c 是从 matlabroot/Simulink/src/目录下的 timestwo.c 变化来的。其具体的变化是将 timestwo.c 中的第一个语句

```
# define S_FUNCTION_NAME timestwo
```

中的 timestwo 变为 doubleinput。这样做的目的有两个：

其一，因为 Simulinkk 还提供了一个名为 timestwo.m 的 M 文件 S－函数，我们知道在 S－Function 模块中的 M 文件 S－函数和 C MEX S－函数的调用形式是一样的，尽管 MATLAB 提供了一个估值顺序来区分两个 timestwo 函数（C MEX S－函数优先执行），但是仍然容易混淆，所以不如改变名称来避免这些。

其二，改变名称后，读者可以自己尝试如何使用 mex 命令来生成 MEX 文件。因为在 matlabroot/toolbox/Simulink/blocks 目录下已经存在了一个编译好的 timestwo.dll 文件。

基于上述的原因，就将 S－函数的名称改为 doubleinput，于是文件名也要相应地保存为 doubleinput.c。

下面就对 doubleinput.c 进行简要的说明：

（1）定义和包含语句。例程首先通过两个定义语句指定恶劣 S－函数的名称（doubleinput）以及该 S－函数属于 level 2 格式。在定义完这两项后，源文件还包含了头文件 simstruct.h，它提供了对 SimStruct 数据结构和 MATLAB 应用程序接口函数的访问。我们知道，M 文件 S－函数中的各个方法都是通过 sys 参量来返回结果的，这些结果由 Simulink 自动完成和不同模型元素的对应。而在 C MEX S－函数中，各个 S－函数方法通过对数据结构 SimStruct 进行存取来完成和 Simulink 交互，读者可以从 doubleinput.c 的各个 S－函数方法的参量定义中看出这一点。为了能访问这个数据结构，就必须包含头文件 simstruct.h。在 doubleipnut.c 中，完成定义和包含的语句为：

```
# define S_FUNCTION_NAME timestwo
# define S_FUNCTION_LEVEL 2
# include "simstruc.h"
```

（2）mdlInitializesSizes 定义了 doubleinputs 中的有关信息。这些信息的定义，都是通过一些由 Simulink 提供的接口函数，对数据结构 SimStruct 进行读写来进行。例如：

```
ssSetNumSFcnParams (S,0);
```

就是将 S－函数的参数个数设置为零（就是需要在 S－Function 模块对话框里设置的参数，相当于 M 文件的额外参数）。Doubleinput.c 包含了大多数接口函数的用途，读者在自己编写过程中，只要把参数值替换一下就可以了。

SimdlInitializeSizes 定义的信息有：

（a）零参数，这意味着 S－函数模块对话框中，S－Function 模块对话框的参数编辑框必须为空。如果它包含任何参数，Simulink 会报告一个参数失配。这通过一个 if 语句将设置的参数个数和实际输入的参数个数进行比较来判断是否是参数匹配，如果失配，则把控制权返回给 Simulink，由 Simulink 报告一个错误信息。请看这个语句：

```
if (ssGetNumSFcnParams (S) ! = ssGetSFcnParamsCount (S) ) {
    return; // 控制权返回给 Simulink，然后，Simulink 将会报告参数失配
```

}

(b) 函数具有一个输入端口和一个输出端口,并且定义输入端口和输出端口的宽度都属于动态可变。对于动态可变的缺省处理是使输入和输出的宽度相同。

完成这输入端口和输出端口个数的设置的接口函数分别是:ssSetNumInputPorts 和 ssSetNumOutputPorts 。在例程中,首先判断这两个函数是否成功地设置了端口个数,如果没有,就把控制权返回到 Simulin。否则,就通过 ssSetInputPortWidth 和 ssSetOutputPortWidth 这两个接口函数分别设置输入端口和输出端口的宽度,宽度值可以是一个具体的整数。在这里由于是动态可变的,所以设置语句分别是:

```
ssSetInputPortWidth ( S, 0, DYNAMICHLLY _SIZED) ;
```

和

```
FssSetInputPortDircetFeedThrough (S,0, 1) ;
```

这里要提醒读者注意,C MEX S-函数中,端口的编号是从 0 开始的,而不是 MATLAB 数组中的从 1 开始。因此,上面的语句中的函数的第二个参量都是 0。

在本例中,由于输出信号直接是将输入信号乘以 2, 所以输入端口是直接馈入的,为此要用 ssSetInputPortDirectFeedThrough 接口函数来设置:

```
ssSetInputPortDirectFeedThrough (S,0,1) ;
```

其中的 0 代表要设置的端口,1 表示具有直接馈入,0 则是没有直接馈入。从这里读者可以看出,在 C MEX S-函数源文件里设定函数的输入输出端口和 M 文件 S-函数的不同,前者是区分端口,它允许用户先设置端口的个数,再来设定每个端口的宽度。

(c)S-函数只有一个采样时间。关于采样时间的设置,C MEX S-函数和 M 文件 S-函数有所不同。C MEX S-函数里,在 mdlInitiallizesSizes 里只能通过 ssSetNumSampleTimes 来指定采样时间的个数,而采样时间的时间值用户必须在 mdlInitializesSampleTimes 方法里指定。

(d)代码被设置为是无异常(exception free)的。指定代码为无异常代码,可以加速 S-函数的执行。当指定这个选项时,一定要小心。一般而言,如果 S-函数不是正在和 MATLAB 交互作用,指定这个选项是安全的。更详细的描述,参看后面的小节。完成这个设置的语句是:

```
ssSetOptions (S, SS_OPTION_EXCEPTION_FREE_DOCE);
```

(3) mdlInitializeSampleTime 方法:

例程通过两条语句分别设置 S-函数的采样时间和偏移时间量大小。它的采样时间从驱动模块继承而来,这意味着 S-函数在接收到输入的任何时候,都好运行。而偏移时间则被设置为 0。

(4) mdlOutput 方法如下所示。

这个方法的作用是进行数值计算,mdlOutput 将输入信号值乘以 2,把结果放到输出信号中。值得注意的有如下几点:

(a)访问输入信号的方法是,使用 ssGetInputPortRealSignalPtrs 接口函数获得一个指向 SimStruct 中某个端口的数据存储区域的指针,并把它赋给一个指针向量。

```
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs (S, 0) ;
```

其中,uPtrs 是一个指针向量,它的数据类型是 InputRealPtrsType,这是 Simulink 自

定义的类型,必须使用下面命令来访问该端口向量的第 i 个元素:

* uPtrs[i]

(b) 访问输出信号的方法和访问输入信号的方法类似,使用的语句是

real _ T * y = ssGetOutputPortRealSignal (S, 0) ;

这返回了模块输出的连续信号。

(c)最后要注意代码中的循环语句。这个 S - 函数可以处理向量输入信号。但是 C 语言不能像 MATLAB 那样直接进行向量运算,它以标量计算为基础,所以要用一个循环次数为信号宽度的 For 循环来处理该端口的所有信号元素。为此,必须首先获得输入和输出端口的宽度,ssGetOutputPortWidth 函数可以获得输出端口的宽度。

(5) mdlTerminate:

这是一种强制性的 S - 函数方法。Timestwo S - 函数不需要实现任何的结束操作,因此这个方法是空的。

(6)在 S - 函数代码的末尾,指定和这个代码关联的代码是 Simulink 还是 Real - Time workshop:

```
# ifdef MATLAB _ MEX _ FILE
# include " Simulink.c'
# else
# include "cg _ sfun.h"
# ednif
```

10.3.3 建立更复杂的 C MEX S - 函数

简单的 C MEX S - 函数只包含一些必需的 S - 函数方法,编写时,可以使用模板 sfuntmpl.c 来简化编写过程。但是如果读者想建立更复杂的 C MEX S - 函数,那么就必须使用更复杂的模板文件 sfuntmpl.c,sfuntmpl.doc 里则有所有可以使用的方法的描述(在 Simulink/src 目录里)。

1.S - 函数顶端要求的语句

为了 S - 函数正常的运行,每一个要访问数据结构的 SimStruct 的 S - 函数源代码块必须包含下面的定义和包含语句。

```
# define S _ FUNCTION _ NAME your - sfunction _ name _ here
# define SFUNCTION _ LEVEL 2
# include "simstruc.h"
```

其中,your _ sfunction _ name _ here 是 S - 函数的名称。这些语句使得 S - 函数可以访问数据结构 SimStruct,它包含了指向仿真中用到的数据的指针。这个 incude 代码同样定义了在 SimuStruct 中存储和获取数据的接口函数。

所谓的 level 1 格式 S - 函数是在 Simulink1.3 到 2.1 建立的,尽管它们和 Simulink3.0 兼容,但是建议最好还是在 level 2 格式下重写 S - 函数。

当 S - 函数被编译为 MEX 文件时,matlabroot/Simulinkv/include/simstruc.h 包含的头文件如表 10 - 5 所示。

表 10－5 被编译为 MEX 文件时 simtruc.h 包含的头文件

头文件	说 明
matlabroot/extern/include/tmwtypes.h	通用的数据类型，例如，vreal_T
matlabroot/extern/include/mex.h	MATLAB MEX 文件 API 方法
matlabroot/extern/include/matrix.h	MATLAB MEX 文件 API 方法

当 S－函数被编译为和 RTW 一起工作时，Simstr.h 包含的头文件如表 10－6 所示。

表 10－6 应用于 RTW 时所包含的头文件

头文件	说 明
matlabroot/extern/include/tmwtypes.n	通用的数据类型，例如，real_T
matlabroot/rtw/c/linstc/rt_matrx.h	MATLAB MEX 文件 API 方法

2. S－函数低端规定的代码

在 S－函数的低端必须具备的代码有：

```
# ifdef MATLAB_MEX_FILE  /* Is this being compiled as MEX - file?  */
# include "Simulin.c"     /* MEX - file interface mechanism */
# else
# include "cg_sfun.h"     /* Code generation registration func */
# endif
```

这些语句为用户选择合适的代码。如果 S－函数被编译为 MEX 文件，则 Simulink.c 文件被包含，如果这个文件是用于和 RTW 结合生成单机或者实时可执行代码时，则 cg_sfun.h 文件被包含。

3. 条件编译 S－函数

S－函数可以被编译成由下面三种定义表示的三种模式之一：

(1) MATLAB_MEX_FILE——表示 S－函数被编译为供 Simulink 使用的 MEX 文件。

(2) RT——表示 S－函数被建立为使用固定步长解法器的实时应用的 RTW 实时代码。

(3) NRT——表示 S－函数被建立为使用可变步长解法器的非实时应用的 RTW 生成代码。

4. 错误控制

当使用 S－函数时，正确地处理意料之外的事件十分重要，例如遇到无效的参数值。在上面的 timestwo 例子中，没有必要使用明显的错误控制。唯一实现错误控制的语句就是 mdlInitializeSizes 方法中的前面出现的 return 语句。那里，处理了一个参数失配的情况，即使用时在对话框输入参数。如果 S－函数具有需要进行有效性验证的参数，可以使用下面的技术来报告遇到的错误：

```
ssSetErrorStatus (S, "error encountered dut to ...") ;
return ;
```

注意 ssSetErrorStatus 的第二个参量必须是永久内存变量，它不能是过程中的局部变

量。例如,下面的代码会导致无法预计的错误:

```
mdlOutputs ()
{
char msg[256_; {ILLEGAL: to fix use "static char msg [256];t"}
sprintf (msg, "Error due to %s", string) ;
ssSetErrorStatus (S, msg) ;
return ;
}
```

ssSetErrorStatus 错误控制方法是 mexErrMsgTxt 函数的推荐替代方法。函数 mexErrMsgTxt 使用异常处理来立即终止 S-函数的执行,并且返回控制给 Simulink。为了在 S-函数里支持异常处理,Simulink 必须在每个 S-函数调用之前安装异常处理器,这会增加仿真的花销。

5. 无异常代码

用户可以通过确信自己的 S-函数只包含无异常代码,来避免这种代价。无异常代码指没有长跳转的代码,有潜在长跳转的代码都不是无异常代码。例如,mexErrMsgTxt 函数调用时,就会产生一个异常结束 S-函数的执行。使用 mexCalloc 将会在发生内存分配错误事件时产生无法预计的结果,因为 mexCalloc 存在长跳转。如果必须进行内存分配,请直接使用 stdib.h 的 calloc 方法并且自己实现错误处理。

如果用户确信函数不会调用 mexErroMsgTxt 或者其他产生异常的 API 方法,那么就使用 SS_OPTION_EXCEPTION_FREE_CODE S-函数选项将代码设定为无异常。这通过 mdlInitializesSizes 函数里的下列命令来实现。

```
SsSetOptions (S, SS_OPTION_EXCEPTION_FREE_CODE);
```

设置这个选项将会使 Simulink 避开通常在每个 S-函数执行前进行的异常处理安装,这就可以提高 S-函数的性能。但是使用这个选项一定要极为仔细,要确保代码是无异常代码。如果在这个选项设置后,S-函数产生了一个异常代码,那么就会发生无法预计的错误。

所有的 mex * 方法都存在长跳转的可能性,此外,几个 mx * 方法也具有长跳转的可能。为了避免产生问题,用户只能使用获得指针和决定参数大小的 API 方法。例如,下面的方法从不会产生异常:mxGetPr, mxGetData, mxGetNumberOfDimensions, mxGetM, mxGetN, mxGetNumberOf Elements。

运行时方法(run-time routines)的代码也能产生异常,运行时方法指 Simulink 在仿真循环中调用的 S-函数方法。运行时方法包括:

(1) mdlGetTimeOfNextVarHit

(2) mdlOutputs

(3) mdlUpdate

(4) mdlDerivatives

如果 S-函数里所有的运行时方法都是无异常的,那么就可以在函数中使用这个选项:

```
ssSetOptions (S,SS_OPTION_RUNTIME_EXECPTION_FREE_CODE) ;
```

而函数中的其他方法则无需是异常的。

10.4 建立C++ S-函数

建立 C++ S-函数是 Simulink4.0 的新增功能,它的过程和建立 C S-函数的过程极为相似,在这一节主要讲述它们之间的区别。

将 C++ S-函数源文件建立为可执行的 S-函数的方法和 C S-函数的方法是相同的,也就是在 MATLAB 命令窗口输入:

> > mex sfun_counter_cpp.cpp

% 将 mex sfun_counter_cpp.cpp 建立为同名的 S-函数。

只不过要注意,源文件的扩展名必须是.cpp,这样使编译器把文件里的代码作为 C++代码来对待。

10.4.1 源文件格式

S-函数的 C++源文件的格式和用 C 编写的 S-函数的源文件几乎完全相同。主要的区别是 S-函数文件必须告诉 C++编译器一些信息,让它在编译调用 S-函数方法时,使用 C 调用规则。这是因为 Simulink 仿真引擎假定所调用的 S-函数方法遵循 C 调用的规则。

要做到这一点,可以通过把 S-函数调用方法的 C++源文件,用一条 extern"C"语句封装起来,sfun_counter 的 C++版本(matlabroot/simulink/src/sun_counter_cpp.cpp)演示了使用 extern "C"语句,使得编译器产生与 Simulink 兼容的调用方法。Sfun_counter_cpp.cpp 的代码如下:

```
/*  File    : sfun_counter_cpp.cpp
 *  Abstract:
 *
 *       Example of an C++ S-function which stores an C++ object in
 *       the pointers vector PWork.
 *
 *  Copyright 1990-2000 The MathWorks, Inc.
 *  $Revision: 1.1 $
 */

#include "iostream.h"

class counter {
    double x;
public:
    counter() {
        x = 0.0;
    }
```

```
    double output(void) {
        x = x + 1.0;
        return x;
    }
};

#ifdef __cplusplus
extern "C" { // use the C fcn-call standard for all functions
#endif       // defined within this scope
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME sfun_counter_cpp

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"

/*========================*
 * S-function methods *
 *========================*/

/* Function: mdlInitializeSizes
==============================================
 * Abstract:
 *     The sizes information is used by Simulink to determine the S-function
 *     block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    /* See sfuntmpl.doc for more details on the macros below */

    ssSetNumSFcnParams(S, 1); /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual parameters */
        return;
    }
```

```
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (! ssSetNumInputPorts(S, 0)) return;

    if (! ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 1); // reserve element in the pointers vector
    ssSetNumModes(S, 0); // to store a C + + object
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}

/*                Function:                    mdlInitializeSampleTimes
=====================================================================
===
 * Abstract:
 *     This function is used to specify the sample time(s) for your
 *     S-function. You must register the same number of sample times as
 *     specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct * S)
{
    ssSetSampleTime(S, 0, mxGetScalar(ssGetSFcnParam(S, 0)));
    ssSetOffsetTime(S, 0, 0.0);

}

#define MDL_START /* Change to #undef to remove function */
#if defined(MDL_START)
/* Function: mdlStart
=====================================================================
=================
  * Abstract:
```

```
 *      This function is called once at start of model execution. If you
 *      have states that should be initialized once, this is the place
 *      to do it.
 */
static void mdlStart(SimStruct * S)
{
  ssGetPWork(S)[0] = (void * ) new counter;   // store new C + + object in the
}                                             // pointers vector
#endif /* MDL _ START */

/* Function: mdlOutputs
 =====================================================================
 * Abstract:
 *      In this function, you compute the outputs of your S - function
 *      block. Generally outputs are placed in the output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct * S, int _ T tid)
{
    counter * c = (counter * ) ssGetPWork(S)[0];    // retrieve C + + object from
    real _ T * y = ssGetOutputPortRealSignal(S,0);  // the pointers vector and use
    y[0] = c - > output();                          // member functions of the
}                                                   // object

/* Function: mdlTerminate
 ====================================================================
 * Abstract:
 *      In this function, you should perform any actions that are necessary
 *      at the termination of a simulation. For example, if memory was
 *      allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct * S)
{
    counter * c = (counter * ) ssGetPWork(S)[0];    // retrieve and destroy C + +
    delete c;                                       // object in the termination
}                                                   // function
/*=========================================================================*
```

```
 * See sfuntmpl.doc for the optional S - function methods *
 * ======================================
=====================*/

/*==================================*
 * Required S - function trailer *
 *==================================*/

#ifdef MATLAB_MEX_FILE          /* Is this file being compiled as a MEX - file?
*/
#include "simulink.c"           /* MEX - file interface mechanism */
#else
#include "cg_sfun.h"            /* Code generation registration function */

#endif
#ifdef _cplusplus
} // end of extern "C" scope
#endif
```

从上面的源文件，读者可以看到定义所有的 S－函数调用方法的源代码，都被包含在 extern"C"语句中，如下所示。

extern "C" {定义调用方法的源代码};

10.4.2 建立永久 C++对象

用户的 C++ S－函数调用方法也许要建立永久 C++对象，也就是在退出方法的执行后也继续存在的对象。例如，一个调用方法可能要访问它的前一次调用中建立的对象，或者是一个调用方法可能要访问另外一个方法所建立的对象。在自己的 S－函数里，建立永久的 C++对象的步骤如下：

(1)建立一个指针工作向量，在方法调用之间保存永久对象的指针。其方法如下：

```
static void mdlInitializesSizes (SimStruct *S)
{
......
ssSetNumPWork (S, 1) ;
// 在指针向量里保留一个元素来存储 C++对象。
......
}
```

其中，ssSetNumPwork 方法的作用是在数据结构 SimStruct 里设置指针工作向量的数目，它的第二个参量，就表示要设置指针工作向量的个数，在上面的代码里是 1。

(2)为要设为永久的每个对象各自存储一个指针到指针工作向量里。例如，

```
static void mdlStart (SimStruct *S)
```

```
{
    ssGetPWork (S) [0] = (void * ) nex counter ;
// 在指针向量 存储新的 C + +对象
}
```

因为在初始化函数里，已经设置了一个指针工作向量，所以在这里，就可以用 ssGetPWork (S) 来获取这个指针数组，因为 C + +数组是从 0 开始计数的，所有数组的下标就是 0。

(3)在后面的方法调用中从指针向量获得原来保存的指针，以访问对象。例如：

```
static void mdlOutputs (SimStruct * S, int _ T tid)
{
    counter * c = ( counter * ) ssGetPWork (S) [0] ;
// 从指针向量获得 C + +对象，并使用对象的成员函数。
real _ T * y = ssGetOutputProtRealSignal (S,0) ;
y[0] = c- > output () ;
}
```

(4) 仿真结束时，销毁对象。例如，

```
static void mdlTerminate (SimStruct * S)
{
  counter * c = (counter * ) ssGetPWork (S) [0];
// 在 mdlTerminate 函数里获得并且销毁 C + +对象
delete c ;
}
```

附录 A　常用“关键符(词)”

[使用说明]

• 本“索引”列出了在本书叙述文字或算例中所涉及的所有符号、指令、模块和图形对象属性的“关键符(词)”。

A.1　MATLAB 的标点及符号

A.1.1　算术运算符 Arithmetic operators

+	加 ;正号
-	减;负号
*	矩阵乘
.*	数组乘
\	矩阵左除
/	矩阵右除
.\	数组左除
./	数组右除
^	矩阵幂
.^	数组幂

A.1.2　关系运算符 Relational operators

= =	等于
~ =	不等于
<	小于
>	大于
< =	小于等于
> =	大于等于

A.1.3　逻辑运算符 Logical operators

&	逻辑与
|	逻辑或
~	逻辑非

A.1.4 特殊符号 Special characters

,	逗号
;	分号
	空格
.	小数点号、构架域号
:	冒号
...	续行号
' '	单引号
.'	转置号
=	赋值号
_	下连符
!	调用 DOS 操作指令号
( )	圆括号
[ ]	方括号
[ ]	空阵
{ }	花括号
%	注释号
%#	编译注记

A.2 MATLAB 的函数及指令 Functions and Commands

A.2.1 A a

abs	绝对值、模、字符的 ASCII 码值
acos	反余弦
acosh	反双曲余弦
acot	反余切
acoth	反双曲余切
acsc	反余割
acsch	反双曲余割
align	启动图形对象几何位置排列工具
all	所有元素非零为真
angle	相角
ans	表达式计算结果的缺省变量名
any	所有元素非全零为真
area	面域图
argnames	函数 M 文件宗量名

asec	反正割
asech	反双曲正割
asin	反正弦
asinh	反双曲正弦
assignin	向变量赋值
atan	反正切
atan2	四象限反正切
atanh	反双曲正切
autumn	红黄调秋色图阵
axes	创建轴对象的低层指令
axis	控制轴刻度和风格的高层指令

A.2.2 B b

bar	二维直方图
bar3	三维直方图
bar3h	三维水平直方图
barh	二维水平直方图
base2dec	X 进制转换为十进制
bin2dec	二进制转换为十进制
blanks	创建空格串
bone	蓝色调黑白色图阵
box	框状坐标轴
break	while 或 for 环中断指令
brighten	亮度控制

A.2.3 C c

capture	(5.3 版以前)捕获当前图形
cart2pol	直角坐标变为极或柱坐标
cart2sph	直角坐标变为球坐标
cat	串接成高维数组
caxis	色标尺刻度
cd	指定当前目录
cdedit	启动用户菜单、控件回调函数设计工具
cdf2rdf	复数特征值对角阵转为实数块对角阵
ceil	向正无穷取整
cell	创建元胞数组
cell2struct	元胞数组转换为构架数组
celldisp	显示元胞数组内容
cellplot	元胞数组内部结构图示

char	把数值、符号、内联类对象转换为字符对象
chi2cdf	χ^2 分布累计概率函数
chi2inv	χ^2 分布逆累计概率函数
chi2pdf	χ^2 分布概率密度函数
chi2rnd	χ^2 分布随机数发生器
chol	Cholesky 分解
clabel	等位线标识
cla	清除当前轴
class	获知对象类别或创建对象
clc	清除指令窗
clear	清除内存变量和函数
clf	清除图对象
clock	时钟
collect	符号计算中同类项合并
colmmd	列最小度排序
colorbar	色标尺
colorcube	三浓淡多彩交叉色图矩阵
colordef	设置色彩缺省值
colormap	色图
colspace	列空间的基
close	关闭指定窗口
colperm	列排序置换向量
comet	彗星状轨迹图
comet3	三维彗星轨迹图
compass	射线图
compose	求复合函数
cond	(逆)条件数
condeig	计算特征值、特征向量同时给出条件数
condest	范 -1 条件数估计
conj	复数共轭
contour	等位线
contourf	填色等位线
contour3	三维等位线
contourslice	四维切片等位线图
conv	多项式乘、卷积
cool	青紫调冷色图
copper	古铜调色图
cos	余弦

cosh	双曲余弦
cot	余切
coth	双曲余切
cplxpair	复数共轭成对排列
csc	余割
csch	双曲余割
cumsum	元素累计和
cumtrapz	累计梯形积分
cylinder	创建圆柱

A.2.4 Dd

dblquad	二重数值积分
deal	分配宗量
deblank	删去串尾部的空格符
dec2base	十进制转换为 X 进制
dec2bin	十进制转换为二进制
dec2hex	十进制转换为十六进制
deconv	多项式除、解卷
delaunay	Delaunay 三角剖分
del2	离散 Laplacian 差分
demo	MATLAB 演示
det	行列式
diag	矩阵对角元素提取、创建对角阵
diary	MATLAB 指令窗文本内容记录
diff	数值差分、符号微分
digits	符号计算中设置符号数值的精度
dir	目录列表
disp	显示数组
display	显示对象内容的重载函数
dlinmod	离散系统的线性化模型
dmperm	矩阵 Dulmage - Mendelsohn 分解
dos	执行 DOS 指令并返回结果
double	把其他类型对象转换为双精度数值
drawnow	更新事件队列强迫 MATLAB 刷新屏幕
dsolve	符号计算解微分方程

A.2.5 Ee

echo	M 文件被执行指令的显示
edit	启动 M 文件编辑器

eig	求特征值和特征向量
eigs	求指定的几个特征值
end	控制流 FOR 等结构体的结尾、数组每维最后元素下标
eps	浮点相对精度
error	显示出错信息并中断执行
errortrap	错误发生后程序是否继续执行的控制
erf	误差函数
erfc	误差补函数
erfcx	刻度误差补函数
erfinv	逆误差函数
errorbar	带误差限的曲线图
etime	计算时间间隔
etree	消去树
etreeplot	画消去树
eval	串演算指令
evalin	跨空间串演算指令
exist	检查变量或函数是否已定义
exit	退出 MATLAB 环境
exp	指数函数
expand	符号计算中的展开操作
expint	指数积分函数
expm	常用矩阵指数函数
expm1	Pade 法求矩阵指数
expm2	Taylor 法求矩阵指数
expm3	特征值分解法求矩阵指数
eye	单位阵
ezcontour	画等位线的简捷指令
ezcontourf	画填色等位线的简捷指令
ezgraph3	画表面图的通用简捷指令
ezmesh	画网线图的简捷指令
ezmeshc	画带等位线的网线图的简捷指令
ezplot	画二维曲线的简捷指令
ezplot3	画三维曲线的简捷指令
ezpolar	画极坐标图的简捷指令
ezsurf	画表面图的简捷指令
ezsurfc	画带等位线的表面图的简捷指令

A.2.6 Ff

factor	符号计算的因式分解
feather	羽毛图
feedback	反馈连接
feval	执行由串指定的函数
fft	离散 Fourier 变换
fft2	二维离散 Fourier 变换
fftn	高维离散 Fourier 变换
fftshift	直流分量对中的谱
fieldnames	构架域名
figure	创建图形窗
fill	二维多边形填色图
fill3	三维多边形填色图
find	寻找非零元素下标
findobj	寻找具有指定属性的对象图柄
findstr	寻找短串的起始字符下标
findsym	机器确定内存中的符号变量
finverse	符号计算中求反函数
fix	向零取整
flag	红白蓝黑交错色图阵
fliplr	矩阵的左右翻转
flipud	矩阵的上下翻转
flipdim	矩阵沿指定维翻转
floor	向负无穷取整
flops	浮点运算次数
flow	MATLAB 提供的演示数据
fmin	求单变量非线性函数极小值点(旧版)
fminbnd	求单变量非线性函数极小值点
fmins	单纯形法求多变量函数极小值点(旧版)
fminunc	拟牛顿法求多变量函数极小值点
fminsearch	单纯形法求多变量函数极小值点
fnder	对样条函数求导
fnint	利用样条函数求积分
fnval	计算样条函数区间内任意一点的值
fnplt	绘制样条函数图形
fopen	打开外部文件
for	构成 for 环用
format	设置输出格式

fourier	Fourier 变换
fplot	返函绘图指令
fprintf	设置显示格式
fread	从文件读二进制数据
fsolve	求多元函数的零点
full	把稀疏矩阵转换为非稀疏阵
funm	计算一般矩阵函数
funtool	函数计算器图形用户界面
fzero	求单变量非线性函数的零点

A.2.7 G g

gallery	特殊测试矩阵
gamma	Γ 函数
gammainc	不完全 Γ 函数
gammaln	Γ 函数的对数
gca	获得当前轴句柄
gcbo	获得正执行“回调”的对象句柄
gcf	获得当前图对象句柄
gco	获得当前对象句柄
geomean	几何平均值
get	获知对象属性
getfield	获知构架数组的域
getframe	获取影片的帧画面
ginput	从图形窗获取数据
global	定义全局变量
gplot	依图论法则画图
gradient	近似梯度
gray	黑白灰度
grid	画分格线
griddata	规则化数据和曲面拟合
gtext	由鼠标放置注释文字
guide	启动图形用户界面交互设计工具

A.2.8 H h

harmmean	调和平均值
help	在线帮助
helpwin	交互式在线帮助
helpdesk	打开超文本形式用户指南
hex2dec	十六进制转换为十进制

hex2num	十六进制转换为浮点数
hidden	透视和消隐开关
hilb	Hilbert 矩阵
hist	频数计算或频数直方图
histc	端点定位频数直方图
histfit	带正态拟合的频数直方图
hold	当前图上重画的切换开关
horner	分解成嵌套形式
hot	黑红黄白色图
hsv	饱和色图

A.2.9 Ii

if – else – elseif	条件分支结构
ifft	离散 Fourier 反变换
ifft2	二维离散 Fourier 反变换
ifftn	高维离散 Fourier 反变换
ifftshift	直流分量对中的谱的反操作
ifourier	Fourier 反变换
i, j	缺省的"虚单元"变量
ilaplace	Laplace 反变换
imag	复数虚部
image	显示图像
imagesc	显示亮度图像
imfinfo	获取图形文件信息
imread	从文件读取图像
imwrite	把图像写成文件
ind2sub	单下标转变为多下标
inf	无穷大
info	MathWorks 公司网点地址
inline	构造内联函数对象
inmem	列出内存中的函数名
input	提示用户输入
inputname	输入宗量名
int	符号积分
int2str	把整数数组转换为串数组
interp1	一维插值
interp2	二维插值
interp3	三维插值
interpn	N 维插值

interpft	利用 FFT 插值
intro	MATLAB 自带的入门引导
inv	求矩阵逆
invhilb	Hilbert 矩阵的准确逆
ipermute	广义反转置
isa	检测是否给定类的对象
iscell	若是元胞数组则为真
iscellstr	若是字符串组成的元胞数组则为真
ischar	若是字符串则为真
isequal	若两数组相同则为真
isempty	若是空阵则为真
isfinite	若全部元素都有限则为真
isfield	若是构架域则为真
isglobal	若是全局变量则为真
ishandle	若是图形句柄则为真
ishold	若当前图形处于保留状态则为真
isieee	若计算机执行 IEEE 规则则为真
isinf	若是无穷数据则为真
isletter	若是英文字母则为真
islogical	若是逻辑数组则为真
ismember	检查是否属于指定集
isnan	若是非数则为真
isnumeric	若是数值数组则为真
isobject	若是对象则为真
isprime	若是质数则为真
isreal	若是实数则为真
isspace	若是空格则为真
issparse	若是稀疏矩阵则为真
isstruct	若是构架则为真
isstudent	若是 MATLAB 学生版则为真
iztrans	符号计算 Z 反变换

A.2.10 J j, K k

jacobian	符号计算中求 Jacobian 矩阵
jet	蓝头红尾饱和色
jordan	符号计算中获得 Jordan 标准型
keyboard	键盘获得控制权
kron	Kronecker 乘法规则产生的数组

A.2.11 L l

laplace	Laplace 变换
lasterr	显示最新出错信息
lastwarn	显示最新警告信息
leastsq	解非线性最小二乘问题(旧版)
legend	图形图例
length	数组长度
light	创建光对象
lighting	照明模式
line	创建线对象
lines	采用 plot 画线色
linmod	获连续系统的线性化模型
linmod2	获连续系统的线性化精良模型
linspace	线性等分向量
ln	矩阵自然对数
load	从 MAT 文件读取变量
log	自然对数
log10	常用对数
log2	底为 2 的对数
loglog	双对数刻度图形
logm	矩阵对数
logspace	对数分度向量
lookfor	按关键字搜索 M 文件
lower	转换为小写字母
lsqnonlin	解非线性最小二乘问题
lu	LU 分解

A.2.12 M m

mad	平均绝对值偏差
magic	魔方阵
maple	运作 Maple 格式指令
mat2str	把数值数组转换成输入形态串数组
material	材料反射模式
max	找向量中最大元素
mbuild	产生 EXE 文件编译环境的预设置指令
mcc	创建 MEX 或 EXE 文件的编译指令
mean	求向量元素的平均值
median	求中位数

menuedit	启动设计用户菜单的交互式编辑工具
mesh	网线图
meshz	垂帘网线图
meshgrid	产生“格点”矩阵
methods	获知对指定类定义的所有方法函数
mex	产生 MEX 文件编译环境的预设置指令
mfun	对 MAPLE 经典函数实施数值计算
mfunlis	能被 mfun 计算的 MAPLE 经典函数列表
mhelp	引出 Maple 的在线帮助
min	找向量中最小元素
mkdir	创建目录
mkpp	逐段多项式数据的明晰化
mod	模运算
more	指令窗中内容的分页显示
movie	放映影片动画
moviein	影片帧画面的内存预置
mtaylor	符号计算多变量 Taylor 级数展开

A.2.13 Nn

ndims	求数组维数
NaN	非数(预定义)变量
nargchk	输入宗量数验证
nargin	函数输入宗量数
nargout	函数输出宗量数
ndgrid	产生高维格点矩阵
newplot	准备新的缺省图、轴
nextpow2	取最接近的较大 2 次幂
nnz	矩阵的非零元素总数
nonzeros	矩阵的非零元素
norm	矩阵或向量范数
normcdf	正态分布累计概率密度函数
normest	估计矩阵 2 范数
norminv	正态分布逆累计概率密度函数
normpdf	正态分布概率密度函数
normrnd	正态随机数发生器
notebook	启动 MATLAB 和 Word 的集成环境
null	零空间
num2str	把非整数数组转换为串
numden	获取最小公分母和相应的分子表达式

nzmax	指定存放非零元素所需内存

A.2.14 O o

ode113	非 Stiff 微分方程变步长解算器
ode15s	Stiff 微分方程变步长解算器
ode23	非 Stiff 微分方程变步长解算器
ode23s	Stiff 微分方程解算器
ode23t	适度 Stiff 微分方程解算器
ode23tb	Stiff 微分方程解算器
ode45	非 Stiff 微分方程变步长解算器
odefile	ODE 文件模板
odeget	获知 ODE 选项设置参数
odephas2	ODE 输出函数的二维相平面图
odephas3	ODE 输出函数的三维相空间图
odeplot	ODE 输出函数的时间轨迹图
odeprint	在 MATLAB 指令窗显示结果
odeset	创建或改写 ODE 选项构架参数值
ones	全 1 数组
optimset	创建或改写优化泛函指令的选项构架参数值
orient	设定图形的排放方式
orth	值空间正交化

A.2.15 P p

pack	收集 MATLAB 内存碎块扩大内存
pagedlg	调出图形排版对话框
patch	创建块对象
path	设置 MATLAB 搜索路径的指令
pathtool	搜索路径管理器
pause	暂停
pcode	创建预解译 P 码文件
pcolor	伪彩图
peaks	MATLAB 提供的典型三维曲面
permute	广义转置
pi	(预定义变量)圆周率
pie	二维饼图
pie3	三维饼图
pink	粉红色图矩阵
pinv	伪逆
plot	平面线图

plot3	三维线图
plotmatrix	矩阵的散点图
plotyy	双纵坐标图
poisscdf	泊松分布概率分布函数
poisspdf	泊松分布累计概率分布函数
poissinv	泊松分布逆累计概率分布函数
poissrnd	泊松分布随机数发生器
pol2cart	极或柱坐标变为直角坐标
polar	极坐标图
poly	矩阵的特征多项式、根集对应的多项式
poly2str	以习惯方式显示多项式
poly2sym	双精度多项式系数转变为向量符号多项式
polyder	多项式导数
polyfit	数据的多项式拟合
polyval	计算多项式的值
polyvalm	计算矩阵多项式
pow2	2 的幂
ppval	计算分段多项式
pretty	以习惯方式显示符号表达式
print	打印图形或 SIMULINK 模型
printsys	以习惯方式显示有理分式
prism	光谱色图矩阵
procread	向 MAPLE 输送计算程序
profile	函数文件性能评估器
propedit	图形对象属性编辑器
pwd	显示当前工作目录

A.2.16 Q q

quad	低阶法计算数值积分
quad8	高阶法计算数值积分
quit	推出 MATLAB 环境
quiver	二维方向箭头图
quiver3	三维方向箭头图

A.2.17 R r

rand	产生均匀分布随机数
randn	产生正态分布随机数
randperm	随机置换向量
range	样本极差

rank	矩阵的秩
rats	有理输出
rcond	矩阵倒条件数估计
real	复数的实部
reallog	在实数域内计算自然对数
realpow	在实数域内计算乘方
realsqrt	在实数域内计算平方根
realmax	最大正浮点数
realmin	最小正浮点数
rectangle	画“长方框”
rem	求余数
repmat	铺放模块数组
reshape	改变数组维数、大小
residue	部分分式展开
return	返回
ribbon	把二维曲线画成三维彩带图
rmfield	删去构架的域
roots	求多项式的根
rose	频数扇形图
rot90	矩阵旋转 90°
rotate	绕指定的原点和方向旋转
rotate3d	启动三维图形视角的交互设置功能
round	向最近整数圆整
rref	简化矩阵为梯形形式
rsf2csf	实数块对角阵转为复数特征值对角阵
rsums	Riemann 和

A.2.18 S s

save	把内存变量保存为文件
scatter	散点图
scatter3	三维散点图
sec	正割
sech	双曲正割
semilogx	X 轴对数刻度坐标图
semilogy	Y 轴对数刻度坐标图
series	串联连接
set	设置图形对象属性
setfield	设置构架数组的域
setstr	将 ASCII 码转换为字符的旧版指令

shading	色彩浓淡模式
shg	使当前图形窗位于前台
shiftdim	数组维序号左移重组
sign	根据符号取值函数
signum	符号计算中的符号取值函数
sim	运行 SIMULINK 模型
simget	获取 SIMULINK 模型设置的仿真参数
simple	寻找最短形式的符号解
simplify	符号计算中进行简化操作
simset	对 SIMULINK 模型的仿真参数进行设置
simulink	启动 SIMULINK 模块库浏览器
sin	正弦
sinh	双曲正弦
size	矩阵的大小
slice	立体切片图
solve	求代数方程的符号解
spalloc	为非零元素配置内存
sparse	创建稀疏矩阵
spconvert	把外部数据转换为稀疏矩阵
spdiags	稀疏对角阵
spfun	求非零元素的函数值
sph2cart	球坐标变为直角坐标
sphere	产生球面
spinmap	色图彩色的周期变化
spline	样条插值
spones	用 1 置换非零元素
sprandsym	稀疏随机对称阵
sprank	结构秩
spring	紫黄调春色图
sprintf	把格式数据写成串
spy	画稀疏结构图
sqrt	平方根
sqrtm	平方根矩阵
squeeze	删去大小为 1 的“孤维”
sscanf	按指定格式读串
stairs	阶梯图
std	标准差
stem	二维杆图
stem3	三维杆图

step	阶跃响应指令
str2double	串转换为双精度值
str2mat	创建多行串数组
str2num	串转换为数
strcat	接成长串
strcmp	串比较
strjust	串对齐
strmatch	搜索指定串
strncmp	串中前若干字符比较
strrep	串替换
strtok	寻找第一间隔符前的内容
struct	创建构架数组
struct2cell	把构架转换为元胞数组
strvcat	创建多行串数组
sub2ind	多下标转换为单下标
subexpr	通过子表达式重写符号对象
subplot	创建子图
subs	符号计算中的符号变量置换
subspace	两子空间夹角
sum	元素和
summer	绿黄调夏色图
superiorto	设定优先级
surf	三维着色表面图
surface	创建面对象
surfc	带等位线的表面图
surfl	带光照的三维表面图
surfnorm	空间表面的法线
svd	奇异值分解
svds	求指定的若干奇异值
switch - case - otherwise	多分支结构
sym2poly	符号多项式转变为双精度多项式系数向量
sym	创建一个符号变量
symmmd	对称最小度排序
symrcm	反向 Cuthill - McKee 排序
syms	创建多个符号对象
symsum	符号计算求级数和

A.2.19 T t

tan	正切

tanh	双曲正切
taylortool	进行 Taylor 逼近分析的交互界面
text	文字注释
tf	创建传递函数对象
tic	启动计时器
title	图名
toc	关闭计时器
trapz	梯形法数值积分
treelayout	展开树、林
treeplot	画树图
tril	下三角阵
trim	求系统平衡点
trimesh	不规则格点网线图
trisurf	不规则格点表面图
triu	上三角阵
try – catch	控制流中的 Try – catch 结构
type	显示 M 文件

A.2.20 U u

uicontextmenu	创建现场菜单
uicontrol	创建用户控件
uimenu	创建用户菜单
unmkpp	逐段多项式数据的反明晰化
unwrap	自然态相角
upper	转换为大写字母

A.2.21 V v

var	方差
varargin	变长度输入宗量
varargout	变长度输出宗量
vectorize	使串表达式或内联函数适于数组运算
ver	版本信息的获取
view	三维图形的视角控制
voronoi	Voronoi 多边形
vpa	任意精度(符号类)数值

A.2.22 W w

warning	显示警告信息
what	列出当前目录上的文件

whatsnew	显示 MATLAB 中 Readme 文件的内容
which	确定函数、文件的位置
while	控制流中的 While 环结构
white	全白色图矩阵
whitebg	指定轴的背景色
who	列出内存中的变量名
whos	列出内存中变量的详细信息
winter	蓝绿调冬色图
workspace	启动内存浏览器

A.2.23 X x, Y y, Z z

xlabel	X 轴名
xor	或非逻辑
yesinput	智能输入指令
ylabel	Y 轴名
zeros	全零数组
zlabel	Z 轴名
zoom	图形的变焦放大和缩小
ztrans	符号计算 Z 变换

A.3 SIMULINK 的库模块

A.3.1 库模块

Demos library	演示子库
Simulink	SIMULINK 基本库

A.3.2 连续模块子库(Continuous)

Continuous	连续模块子库
Derivative	求导数模块
Integrator	连续函数积分
Memory	记忆模块
State – Space	状态方程模块
Transfer Fcn	传递函数模块

A.3.3 离散模块子库(Discrete)

Discrete	离散模块子库
Discrete Filter	离散滤波器模块
Discrete – Time Integrator	离散时间积分模块

Discrete Transfer Fcn　离散传递函数模块
Discrete Zero - Pole　离散零极点增益模块
Unit Delay　单位延迟模块
Zero - Order Hold　零阶保持模块

A.3.4　解析函数和查表函数模块子库(Functions & Tables)

Fcn　C语言格式的任意函数模块
Functions & Tables　解析函数和查表函数模块子库
MATLAB Fcn　MATLAB语言格式的任意函数
Look - Up Table　一维查表函数模块
Look - Up Table(2 - D)　二维查表函数模块

A.3.5　一般数学函数子库(Math)

Abs　取绝对值模块
Combinatorial Logic　组合逻辑模块
Gain　增益模块
Logical　逻辑运算模块
MinMax　取极大值或极小值的模块
Math　一般数学函数子库
Mux　复用模块
Product　乘法器
Relational　关系运算模块
Sign　符号取值模块
Slider　滑键增益模块
Sum　求和模块

A.3.6　非线性模块子库(Nonlinear)

Dead Zone　死区非线性模块
Nonlinear　非线性模块子库
Relay　继电器非线性模块
Saturation　饱和非线性模块

A.3.7　信号和系统模块子库(Signal & Systems)

Demux　分用模块
Enable　使能模块
Ground　接地模块
In1　输入端口模块
Merge　汇合模块
Out1　输出端口模块

Signal & Systems	信号和系统模块子库
SubSystem	子系统模块
Trigger	触发模块
Terminator	终端模块

A.3.8 信宿模块子库(Sinks)

Display	数值显示模块
Scope	示波模块
Sinks	信宿模块子库
Stop	终止仿真
To File	把数据保存为文件
To Workspace	把数据写成为矩阵变量
XY Graph	显示 X－Y 图形

A.3.9 信源模块子库(Sources)

Clock	连续仿真时钟模块
Constant	恒值输出模块
From File	从文件读数据
From Workspace	从内存读数据
Pulse	脉冲发生器
Signal Generator	信号发生器
Sine Wave	正弦波输出
Sources	信源模块子库
Step	阶跃输出

A.4 图形对象的属性

A.4.1 低层对象创建指令和句柄获取指令

axes
figure
findobj
gca
gcbo
gcf
light
line
patch
rectangle

surface
text
uicontrol
uicontextmenu
uimenu

A.4.2 图形对象属性

Accelerator
Ambient
AmbientStrength
BackgroundColor
BackingStore
Box
Callback
CameraPosition
CameraPositionMode
CameraTarget
CameraTargetMode
CameraUpVector
CameraUpVectorMode
CameraViewAngle
CameraViewAngleMode
CData
CDataMapping
Checked
Children
Color
Colormap
ColorOrder
CurrentAxes
CurrentFigure
CurrentPoint
Curvature
DataAspectRatio
DefaultUicontrolUnits
DefaultUicontrolFontname
DefaultUicontrolFontsize
DefaultUicontrolHorizontal
DiffuseStrength

EdgeColor
EdgeLighting
Enable
EraseMode
FaceColor
FaceLighting
Faces
FaceVertexCData
FileName
FontName
FontAngle
FontSize
ForgroundColor
HandleVisibility
Horizontal
HorizontalAlignment
Label
LineStyle
LineWidth
ListboxTop
Marker
MarkerEdgeColor
MarkerFaceColor
MarkerSize
Max
MenuBar
MeshStyle
Min
Name
NextPlot
NumberTitle
PaperPosition
PaperUnits
Parent
PlotBoxAspectRatioMode
Pointer
PointerShapeCData
PointerShapeHotSpot
Position

Projection
Rotation
Separator
SpecularColorReflectance
SpecularExponent
Speculars
SpecularStrength
String
Style
Tag
Title
ToolBar
Type
Units
UserData
Value
VerticalAlignment
Vertices
Visible
WindowButtonDownFcn
WindowButtonMotionFcn
WindowButtonUpFcn
XColor, YColor, ZColor
XData, YData, ZData
XGrid, YGrid, ZGrid
XLabel, YLabel, ZLabel
XLim, YLim, ZLim
XTick, YTick, ZTtick
XTickMode, YTickMode, ZTickMode
XTickLabel, YTickLabel, ZTickLabel
XTickLabelMode, YTickLabelMode, ZTickLabelMode
XScale, YScale, ZScale
XDir, YDir, ZDir
XAxisLocation, YAxisLocation, ZAxisLocation

附录 B Internet 资源

访问 Internet,不论是全面存取或发送简单的电子邮件,MATLAB 有可以利用的丰富资源。通过从教育机构、政府部门及商业公司直接与 Internet 相连,以及通过商业的 Internet 所提供的咨询和在线服务,如:America Online、Delphi、CompuServe 等等,都可获得这些资源。

1. USENET 新闻组

在 Internet 上一个最重要的信息资源和论坛就是一个称之为 NetNews FeeD、USENET 或简称为 News 的公告栏系统。它是一个巨大的消息或公告的集合,在全球范围内从一台计算机到另一台计算机流动。News 是由成千上万的单个的新闻公告组成,它收集在数以千计的新闻组或标题中。许多站点建立了可供利用的所有新闻组,而其他一些地方由于磁盘空间的限制或公司策略关系则更具选择性。

如果访问 NetNews,就可以阅读 MATLAB 的其他用户发出的新闻公告或者发布你自己的评论或问题。MATLAB 新闻组称为 comp.soft - sys.matlab。订阅这个新闻组,可以通过提问、评论和从其他用户以及从 Mathworks 职员来的解答、浏览新闻。实际上,MATLAB 的开发者经常向论坛发布消息。如果在用户站点上得到的新闻组有限,不能找到 comp.soft - sys.matlab,就请与当地的新闻管理人员联系,要求将 comp.soft - sys.matlab 包括进来。

要知道,这个新闻组是未作修饰的,而这意味任何人都可以发布自己的提问或评论,无法将不适当的公告过滤,全世界有数以千计的人阅读这个新闻组,所以发布时要慎重和适当。

如果不能访问 NetNews,但具有 FTP 功能,则如下面讨论的,可以从匿名 FTP 站点上得到这个新闻的公告摘要,即用目录/pub/doc/cssm - digest/中的 ftp.mathworks.com。

如果没有 News,且 FTP 不可选,但有连接到 Internet 的电子邮件(E - mail),在 Mathworks 有一个邮件 - 新闻的网关,允许通过 E - mail 来向新闻组发布消息。由 E - mail 发往 comp. soft - sys. matlab@mathworks. com 的消息将发布到新闻组上。如果你不能阅读新闻组,则一定要在发布的消息中加入请求,使回答直接用 E - mail 送回给你,同时要提供 E - mail 地址。

2. 匿名 FTP

MATLAB用户的一个最有用的资源是由 Mathworks 维护的一个匿名 FTP 站点。FTP 是文件传输协议,用来连接主机,从而将文件送到计算机和从计算机传出。匿名 FTP 站点允许用户和它们相连并传送文件,不要求用户在主机上有账户和口令。

有几个计算机程序用用户图形界面进行 FTP 连接和传送文件。包括 PC 上的 WinFTP,Macintosh 上的 Fetch,Unix 上的 ftptool。如下面要讨论的,Web 浏览器,如 Mosaic,

Netscape 也用于访问这个站点。如果站点没有这些程序，就必须使用原始的面向命令行的 ftp 程序，因为原始 ftp 程序通常在 Unix 工作站上可获得，PC 机和 Macintosh 上也有。它的用法将在本节进行描述。

MATLAB FTP 站点在 ftp.mathworks.com，它包括用户提供的 M 文件、产品信息、含有常问问题(FAQs)、补片和诊断的文档。为了连接到站点，即 ftp 到站点，用 'anonymous' 名字或' ftp ' 登录，然后当要求口令时，输入用户的 E-mail 地址。

下面是一个简短的 ftp 会话示例。它连接到站点，列出可用的文件及目录，变换成 ASCII 模式以传送文本文件；检索文件 README；关闭连接。

```
ftp ftp.mathworks.com
Connected to ftp.mathworks.com
220 ftp FTP server (Version wu-2.r(1) Thu April 14:25:23) ready
Name (ftp.mathworks.com:user):anonymous
331 Guest login ok, send your complete E-mail address as password.
Password: user@host.maine.edu
230
230Welcome to the Mathworks Library!
...
230 Guest lgoin ok, access restrications apply.
Remote system type is UNIX
Using bianry mode too transfer files
ftp> dir
200 PROT command suceessful
150 Opeing ASCII mdoe data connection for /bin/ls
226 Transfer complete.
total 27
-rx-r--r--     1  admin  daemon     488   Jun  24  1994
.welcome.msg
-rw-r--r--     1  admin  ftpusers   2172  Sep  28  1994
README
-rw-r--r--     1  admin  ftpusers   2626  Sep  28  1994
README.incoming
drwxr-xr-x   2  root   daemon    512  Jun  16  1994  bin
drwxr-xr-x   2  toot   daemon    512  Jun  28  1993  dev
drwxr-xr-x   2  toot   daemon    512  Jun  7   1993  etc
drxwrwx-wx  38  edmin  102     14336 Apr' 21  17:23  incoming
lrxwrwxrwx   1  root   daemon      3  Nov  6   1993  matlab->pub
drwxr-xr-x  12  admin  ftpusers  512  Mar  27  15:17  pub
drwxr-xr-x   3  root   daemon    512  Apr  23  1993  usr
```

```
ftp> ascii
200 Type  set to A
ftp> get README
200 port command successful
150 Opening ASCII mdoe data connection for README (2172 bytes).
226 bytes received
ftp> bye
221 Goodbye
```

第 1 列用 ' d ' 列出的是目录，第 1 列有 ' - ' 是文件，第 1 列为 ' l ' 表示连接。它可以指向另外一个目录或其他某站点上的文件。目录列表提供的信息还包括拥有者、组、文件或目录大小、修改日期、文件或目录名称。

如果在连接时出现问题，也许是还没有访问到一个 Internet 名字服务器，该服务器将主机的名称翻译成 IP 地址。在这种情况下，试用 IP 地址 144.212.100.10 来代替 ftp.mathwoeks.com

```
ftp 144.21.100.10
connected to ftp.mathworks.com
```

fpt 程序支持大量的命令和选项。最有用的列在附表 - 1。

附表 - 1　基本 FTP 命令

help	列出所有可用的 ftp 命令
help command	返回一个简单的命令描述
cd /pub/contrib	改变到远程站点上的/pub/contrib 目录
lcd localdir	改变到用户的本地机上的目录
ascii	以 ASCII 文本模式传输文件。对自已机器上的文本文件，将行结束回车/换行变换为缺省值
binary	以二进制方式传输，不进行文本转换
get remotefile	从远程站点检索名为 remotefile 的文件
put localfile	将名为 locatefilede 的文件传送到远程站点
dir	在远程站点列出详细的当前目录
Ls	在远程站点列出当前目录，详细有限
bye, quit	退出 ftp

MATLAB 具有其他许多匿名 FTP 上都不具备的一个非常好的特性，它有能力即时地完成文件的归档和压缩。所支持的文件压缩格式有 MIT 的 GNU 中的 gzip (file.gz)，标准 Unix 上的 compress (file.z)以及在 PC 和 Unix 上的 zip (file.zip)。文件的归档用标准的 Unix tar 格式(dir.tar)、Unix shar 格式(dir.sh)以及 PC 和 Unix 工作站的 zip 格式(dir.zip)，其中各格式对 PC 和 Unix 工作站都可用，且绝大部分也可以用于 Macintosh 计算机。

归档和压缩可以同时进行。如用以下命令，目录 /pub/contrib/math 的全部内容可以在 Unix 压缩的 tar 格式文件中进行检索，

cd /pub/contrib

get math.tar.z

其他的组合也可以。命令

get math.zip

get math.sh

get math.tar.gz

分别以 zip 文档、shar 文档、gzipped tar 文档来检索目录。不幸的是，对于 Macintosh 的用户，MATLAB 站点不支持 stuffit (dir.sit)文档或 BinHex (file.hqx)编码。

附表-2 是 MATLAB 站点的特性。

附表-2 MATLAB 匿名 FTP 站点上重要的文件和目录

名称	类型	内容
/README	文本	新用户的文本信息
/README.incoming	文本	用户提交的文本信息
/incoming	目录	M 文件用户意见投入箱的目录
/pub	目录	文件的主目录
/matlab	连接	连接到 /pub
/pub/INDEX	文本	有关目录内容的信息
/pub/NEWFILES	文本	站点上的新文件
/pub/ls-lr	文本	站点上所有文件的递归列表
/pub/ftphelp	文本	新 ftp 用户的入门指导
/pub/books	目录	与 MATLAB 基础有关的 M 文件
/pub/conference	目录	即将要召开的 MATLAB 会议的信息
/pub/contrib	目录	用户提供的 M 文件和 MEX 文件
/pub/doc	目录	文档，指导，帮助文件，FAQs，等等
/pub/mathworks	目录	MATLAB 开发者提供的诊断和 M 文件
/pub/mosaic	目录	用于 Unix 的 Mosaic 的 WWW 浏览程序
/pub/pentium	目录	奔腾迷论文集
/pub/proceedings	目录	过去的 MATLAB 会议的论文集
/pub/product-info	目录	MATLAB，SIMULINK 和工具箱的信息
/pub/tech-support	目录	技术支持和其他信息

如果有连接的麻烦或需要帮助，可以向 fptadmin@mathworks.com 站点的维护人员发 E-mail。

3. 万维网(WWW)

向 Internet 获取信息、图像、文件的最新、最容易、也是最灵活的方法是 World Wide Web 或简称为 WWW 或 Web。它是根据请求向本地计算机提供超文本文档的站点服务器的集合。超文本文档可以与内置的图形结合在一起并连接到其他文件，只需用鼠标点

击闪亮的文字或图形，就可以激活这些连接。然后，检索所连接的文档进行观察。如果愿意，也可以将文件存到自己的计算机上。有些连接可以有接到 Gopher 站点上（基于文本信息的服务器）以及全球的 FTP 站点。

如果有访问全面服务的 Internet 节点，就有这类 MATLAB 信息的图形界面供选择。Mathworks 有一个 WWW 服务器，可以用 Web 的客户程序如 Mosaic、Netscape 或基于字符的 Lynx 来访问。用菜单项的 Open Location…或 Open URL…然后输入 http:// www. mathworks. com 则可以连接到 Mathworks 的主页。主页是最高层的超文本文档，它包含了到其他的文档的连接。一旦连通，就可以将访问到的信息作为书签保存下来，以方便下一次的连接。

从 Mathworks 的主页，超文本连接可查询 Mathworks 公司和它的产品，在线拷贝通讯季刊，列出 MATLAB 的书籍（在写这本书时已超过 100 本）及相关 M 文件，职员要参加的有关商贸展览会的信息和最新的 MATLAB 会议论文集。还有一个常问问题及其关于 MATLAB 和 SIMULINK 解答的文档库。也可以阅读由 MathWork 技术支持人员提供的技术要点，包括内存管理、图形打印、MEX 文件、工具箱以及如何将 MATLAB 与其他的软件集成的技术和技巧。

Mathworks 主页也有到匿名 FTP 站点上的直接连接，可以浏览目录、读索引文件、并用浏览器检索文件。如果没有 Web 浏览器程序，在 FTP 站点的/pub/mosaic 目录下可以获得许多工作站的 Mosaic 的版本；用于 PC 及 Macintosh 的 Mosaic 也可以通过匿名 FTP 的 ftp.ncsa.uiuc.edu 得到，而用于工作站和微机的 Netscape 可以通过 ftp.netscape.com 得到。必须明白，虽然 Mosaic 是免费 软件，但对免费使用 Netscape，商业公司有一些限制。

4. MATLAB 的自动电子邮件自动应答系统

对于那些不具有 FTP 功能，而有 E-mail 连接到 Internet 上的用户并没有排斥在外。Mathworks 拥有一个自动的用 mail 的文件服务，称为 MATLIB，它从 FTP 站点用 E-mail 传输文件。向 matlab@mathworks.com 发一个 E-mail，在消息主体中带有单向 help 就可以得到有关访问该服务器的更详尽的信息。MATLIB 的 mail 文件服务是一个可以从消息主体中取得其指令信息的计算机程序，标题行被忽略掉。所以，消息句法正确十分重要。

在消息中可以用的许多指令是标准 FTP 命令的子集，包括：cd，ls，dir，get，quit。其他的命令用于指定编码、存档、所用的压缩方式。其他还有指导 matlib 的程序，以回复不同的 E-mail 地址或限制回复文件的大小。

一个例子是 reply-to 命令，这是可选的。但如果用了，就可以使 MATLIB 服务器向 E-mail 地址发送由命令指定的文件。如果没有用，则向发送者的 E-mail 地址回复。例如：

reply-to user@host.maine.edu

则向 user@host.maine.edu 回答，而不是原来的 E-mail 地址。

二进制的文件不能用 E-mail 传送，所以在发送前要编码（转换成 ASCII 字符表示）。二进制文件按缺省是用 uuencode 发送。如果需要，也可以用 mime 编码来发送，也有用 compress、gzip、zip、shar、tar 进行压缩或归档。例如：考虑发送到 matlib@mathworks.com

的一则消息，使用如下命令：

```
dir
cd /pub/contrib
get INDEX
get intergration.tar.z
get math.zip
cd games
get fifteens.m.gz
cd ..
get diffeq.sh
quit
```

这则脚本指示 MATLIB 首先检索目录，列出顶层目录。然后改到/pub/contrib 目录，检索 INDEX 文件。然后作为受压缩的 tar 文件检索整个/pub/contrib/integration/目录，这是二进制文件，所以在传送之前先要编码 uuencoded。接下来对整个/pub/contrib/math 目录的内容，检索 uuencoded zip 文档，现在改到 games 目录，用 gzip 压缩检索 M 文件 fifteens.m，最后改到下一个较高层目录/pub/contrib，作为 shar 文档，检索整个 diffeq 目录，然后退出。

回复限制在 100K 字节以内，所以较长的文件或文档要以多个 E－mail 的方式传送。如果 E－mail 大小有限制，也可以用 size 命令进一步限制消息的大小。help 消息中有更详尽的说明。

5. MathWorks MATLAB 文摘

MATLAB 文摘是电子通报月刊，通过 E－mail 向订户发送。这个通报包括 MATLAB 的新闻，MATLAB 的开发者和用户提供的文章，提示以及对用户问题的回答。想要订阅文摘，则要向 subscribe@mathworks.com 发送 E－mail，申请加入邮寄名单，或在 MATHLAB 提示行中键入 > > subscribe。subscribe 命令提出问题，利用回答产生打印的申请表，该表邮寄或传真到 Mathworks。Unix 提供了自动的对申请发送 E－mail 的功能，而不是打印出来。在 MATLAB FTP 的/pub/doc/tmw－digest 目录中，有过期的通报。

6. MATLAB 通报

使用 MATLAB 任何用户，都可得到季度出版物 MATLAB News & Notes。如果你是订户，可以收到文摘及季度通报以及免费的技术支持。不需注册就可以成为一个订户，这是免费服务的。可利用 WWW 站点上的表格，键入 > > subscribe，或向 subscribe@mathworks.com 发送 E－mail 提出要求。通报通常有新闻、提示、克利夫·莫勒(Cleve Moler)和其他人的文章以及大事表。

7. MathWorks 电子邮件及网络地址

附表－3　Mathworks 公司的 E－mail 地址

suport@mathworks.com	技术支持
bugs@mathworks.com	错误报导
doc@mathworks.com	文档错误报导

(续)

suggest@mathworks.com	产品升级建议
service@amthworks.com	订购情况,延长许可,许可码
subscribe@mathworks.com	订户信息
info@mathworks.com	销售,价格和一般信息
micro-updates@mathworks.com	PC 及 Macintosh 的升级
matlib@mathworks.com	mail 文件服务器
digest@mathworks.com	MATLAB 文摘
ftpadmin@mathworks.com	Mathworks FTP 站点
webmaster@mathworks.com	Mathworks 的 WWW 维护人员

附表-4 MATLAB 的网络资源

www.mathworks.com	WWW 站点
ftp.mathworks.com	匿名 FTP 站点
144.212.100.10	WWW 及 FTP 的 Internet 地址
novell.felk.cvut.cz	ftp.mathworks.com 的镜象
192.108.154.33	Internet 地址的镜象

附表-5 其他的网络资源

mm@eeece.maine.edu	向作者提出有关精通 MATLAB 及精通 MATLAB 工具箱的问题和评论的 E-mail 地址
ftp.ncsa.uiuc.edu	Telnet 和 FTP 的 PC 及 Macintosh 版本,PC、Macintosh 和 Unix 的 Mosaic
ftp2.netscape.com	PC Mac. Unix 的 Netscape 的浏览器
sumex-aim.stanford.edu	Mosaic、Telnet(FTP)、Fetch、gzip、zip、shar、compress 的 Mac 版本
sics.se	在 sumex-aim.stanford.edu 的文件镜象
gatekeeper.dec.com	大的匿名 FTP 站点
sunsite.unc.edu	大的匿名 FTP 站点
ftp.wustl.edu	大的匿名 FTP 站点
nic.funet.fi	大的匿名 FTP 站点
ftp.luth.se	大的匿名 FTP 站点
ftp.cdrom.com	大的匿名 FTP 站点
ftp.nws.edu.au	大的匿名 FTP 站点